ANSI COBOL

INTRODUCTION TO COMPUTER PROGRAMMING

Gary B. Shelly

Thomas J. Cashman

INTRODUCTION TO
COMPUTER PROGRAMMING
ANSI COBOL

ANAHEIM PUBLISHING COMPANY

Specialist In Data Processing Textbooks

INTRODUCTION TO DATA PROCESSING

The Computers In Our Society, Logsdon & Logsdon

ASSEMBLER LANGUAGE

IBM System/360 Assembler Language, Cashman & Shelly
IBM System/360 Assembler Language Workbook, Cashman & Shelly
IBM System/360 Assembler Language Disk/Tape Advanced Concepts, Shelly & Cashman
DOS Job Control For Assembler Language Programmers, Shelly & Cashman

COBOL

Introduction To Computer Programming ANSI COBOL, Shelly & Cashman
ANSI COBOL Workbook, Testing & Debugging Techniques & Exercises, Shelly & Cashman
Advanced ANSI COBOL Disk/Tape Programming Efficiencies, Shelly & Cashman
Introduction To Computer Programming System/360 COBOL, Cashman
IBM System/360 COBOL Problem Text, Cashman
IBM System/360 COBOL Disk/Tape Advanced Concepts, Shelly & Cashman
DOS Job Control For COBOL Progammers, Shelly & Cashman

RPG

Introduction To Computer Programming RPG, Shelly & Cashman
IBM System/360 RPG Programming, Volume 1-Introduction, Fletcher & Cashman
IBM System/360 RPG Programming, Volume 2-Advanced Concepts, Fletcher & Cashman

FORTRAN

Introduction To Computer Programming Basic FORTRAN IV-A Practical Approach, Keys

PL/I

Introduction To Computer Programming System/360 PL/I, Shelly & Cashman

FLOWCHARTING

Introduction To Flowcharting and Computer Programming Logic, Shelly & Cashman
Basic Logic For Program Flowcharting and Table Search, Jones & Oliver

JOB CONTROL-OPERATING SYSTEMS

DOS Utilities Sort/Merge Multiprogramming, Shelly & Cashman
OS Job Control Language, Shelly & Cashman

CERTIFICATE IN DATA PROCESSING

Review Manual For Certificate In Data Processing, Cashman

UNIT RECORD TECHNIQUES

Basic Projects In Data Processing, Cashman & Keys
Practical Projects In Data Processing, Cashman & Keys
Introduction To Control Panel Wiring-548 Interpreter, Cashman & Keys
Control Panel Wiring-514 Reproducer, Cashman & Keys
Control Panel Wiring-85 Collator, Cashman & Keys
Control Panel Wiring-402 Accounting Machine, Cashman & Keys
Control Panel Wiring-407 Accounting Machine, Cashman & Keys

INTRODUCTION

TO

COMPUTER PROGRAMMING

ANSI COBOL

By:

Gary B. Shelly
Educational Consultant
Instructor, Long Beach City College

&

Thomas J. Cashman, CDP, B.A., M.A.
Long Beach City College
Long Beach, California

ANAHEIM PUBLISHING COMPANY
1120 E. ASH
FULLERTON, CALIFORNIA

First Printing
April 1973

Second Printing
September 1973

Third Printing
February 1974

Fourth Printing
June 1974

Fifth Printing
November 1974

Printed in the United States of America

ISBN 0-88236-103-1

© *Copyright 1973 Anaheim Publishing Company*

All rights reserved. This book or any part thereof may not be reproduced without the written permission of the publisher.

PREFACE

The programming language COBOL has taken its place as the dominant language in business programming. Through the efforts of various committees, the standardization of COBOL took place and today, ANSI COBOL is more and more becoming the language being used. The primary attraction of ANSI COBOL is that it is as machine-independent as is possible for a programming language. Thus, the ANSI COBOL taught in this text is applicable to the IBM System/360, the IBM System/370, the IBM System/3, the IBM 1130, NCR Century Series, the UNIVAC series of machines, the CDC hardware, and other manufacturers of computer systems.

This book is designed to serve as an Introductory textbook in computer programming. No prior programming experience is assumed, and all basic concepts including business programming logic, software, and programming are included.

This textbook uses the "problem-oriented" approach. The student is introduced to programming concepts by the means of a series of programs illustrating typical applications. Only those statements and segments of the language necessary in the solution of the problem are explained. The student, therefore, learns ANSI COBOL programming in relation to the total problem, and is not burdened with the task of remembering a series of isolated facts concerning the individual segments of the language. Each of the chapters in the text introduces additional programming concepts. Upon completion of the text, the student will have been exposed to the most frequently used characteristics of the language and should have the capability of writing a wide variety of COBOL programs for the solution of typical business applications.

Every effort has been made to simplify the concepts presented in the text through the use of realistic problems, numerous examples and illustrations, and completely documented and fully programmed examples. Each of the programs illustrated has been written using the ANSI COBOL Compiler and has been fully tested and run on a System/360. The programs explained in each of the chapters include the following operations or programming techniques: basic input/output, addition, subtraction, multiplication, division, compute statement, taking a final total, comparing, heading and print control, minor total logic, major, intermediate, and minor total logic, table lookup, the SORT Verb and table lookup using Set and Search verbs, and reading and writing on sequential disk files. At the conclusion of each chapter, two programming assignments are included. The first of the assignments in each chapter is relatively close to the program presented within the chapter. The second assignment requires the student to apply those principles learned within the chapter to a more challenging problem. It should be noted also that this text is accompanied by a workbook ANSI COBOL WORKBOOK TESTING AND DEBUGGING TECHNIQUES AND EXERCISES, which contains many problems relating to both compilation and execution problems and is a valuable guide to the serious COBOL student.

A comprehensive set of appendices is included providing a convenient reference for the technical aspects of the language. The appendices include such topics as a COBOL Format Summary, a COBOL Reference Summary, Reading a COBOL Listing, Internal Data Representation, and Environment and Procedure Division Entries for Various Machines. Including these topics in the appendices offers the instructor flexibility in presenting these topics in the course without disturbing the logical sequence of the presentation of the text material. It is suggested, however, that the contents of the appendices be included as a formal part of the course of study, and that the student using the text be thoroughly familiar with the concepts presented upon conclusion of the course in ANSI COBOL programming.

After the student has completed the study of the material contained in the text, he should have a firm foundation of the basic concepts of COBOL programming and be capable of solving a wide variety of business programming problems. This book may be effectively used in a one semester course in computer programming at junior colleges, colleges, or private or vocational schools teaching COBOL programming. It should be noted that the problem-oriented approach utilized in this text has been used successfully as a "home study" or self-teaching method and has proven superior to traditional Programmed Instructional approaches.

Gary B. Shelly

Thomas J. Cashman

Acknowledgment

The following information is reprinted from COBOL Edition 1965, published by the Conference on Data Systems Languages (CODASYL).

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgment of the source, but need not quote this entire section.

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

"No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals of similar publications".

TABLE OF CONTENTS

Chapter 1 Introduction to Programming 1.1

Chapter 2 Introduction to Computer Programming
Input/Output Operations 2.1

Chapter 3 Addition – Subtraction 3.1

Chapter 4 Multiplication – Division 4.1

Chapter 5 Compute Statement Final Totals 5.1

Chapter 6 Comparing 6.1

Chapter 7 Condition-Names . . . Go To Options 7.1

Chapter 8 Headings – Print Overflow 8.1

Chapter 9 Minor Total Logic 9.1

Chapter 10 Major, Intermediate, Minor Totals 10.1

Chapter 11 Table Lookup 11.1

Chapter 12 Sort Verb Set/Search Table Lookup 12.1

Chapter 13 Additional COBOL Statements 13.1

Appendix A Internal Data Representation A.1

Appendix B COBOL Format Notation B.1

Appendix C COBOL Reference Summary C.1

Appendix D Entries for Various Manufacturers D.1

Appendix E Reading Source Listing and Diagnostics E.1

Appendix F Test Data F.1

CHAPTER 1

INTRODUCTION TO PROGRAMMING

INTRODUCTION

The data processing industry has undergone dramatic changes since it first began in the late 1940's. Much of the drama has centered around the changes and improvements in the computer hardware and the related peripheral equipment such as card readers, magnetic tape units, disk storage devices, and other input/output units which are used with the computer.

As computer hardware has gone through an evolutionary period, so too has the art or science of programming. This includes both the jobs done by programmers and the methods available to the programmer for solving problems. Programming may be broken down into two broad categories: Systems programming and Applications programming. The systems programmer is concerned with writing programs that make it easier to operate and program computers. Systems programmers turn out control programs that operate input/output equipment, test programs that detect errors and malfunctions within the computer, utility programs, such as sorts, which provide for easy manipulation of data, and programming languages and compilers which are available for the applications programmer to use to solve problems.

Applications programming may be broken down into two general areas: Scientific and Engineering programming and Business programming. The scientific applications programmer is normally involved in programs requiring a great deal of complex mathematical work. In many cases, scientific applications programmers are also mathematicians or engineers who are using the computer to solve problems with which they are directly concerned.

The business applications programmer writes programs to solve problems relating to the business transactions of a company. Such applications as payroll, billing, and inventory control are the areas of concern to a business applications programmer. Business programming is normally not performed by a person such as an accountant or a production manager. Rather, in most businesses, the job of programming is assigned to an individual working in the data processing department. The business programmer writes a variety of programs for various business applications within a company.

BASIC PROGRAMMING CONCEPTS

Even though the types of programming may vary to a large extent, the task of the programmer is always the same, that is, write a program, or set of instructions, which direct the computer to process data in a way which will solve the given problem. The usefulness of computers stems from their ability to perform logical and arithmetic operations accurately and at very high speeds. The number and complexity of the basic operations that can be performed by a given machine varies from computer to computer, but all computers must have some type of program which consists of a series of instructions directing the operations of the computer.

An instruction which causes a computer to perform a given function consists of an operation code, which directs the computer to perform a specific operation, and one or more Operands which are normally used to reference where the data to be processed is stored. When the instruction is executed by the computer, the central processing unit interprets the operation code and the operand as a function to be performed. This concept of placing instructions in the storage unit of the computer to direct it to perform a specific operation is called the "Stored Program" concept.

The set of specific basic operations that can be performed by a given computer is called its instruction set. A portion of the instruction set of the System/360 computer is illustrated below.

Instruction	Machine Operation Code
Add Decimal	FA
Compare Decimal	F9
Divide Decimal	FD
Edit	DE
Edit and Mark	DF
Multiply Decimal	FC
Subtract Decimal	FB
Zero and Add	F8

Figure 1-1 Portion of the System/360 Instruction Set

Note from the example in Figure 1-1 that the operation code of the instruction to cause two numbers to be added is FA. Thus, the operation code "FA", when recorded in the storage unit of the computer and analyzed by the controlling electronic circuitry, will cause the computer to perform an addition operation.

As noted, an instruction which causes an operation to be performed normally consists of an Operation Code as illustrated in Figure 1-1 and one or more operands. The operands used with the addition instruction are illustrated below.

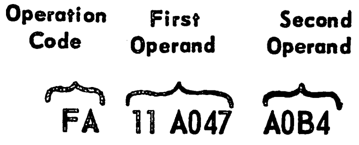


Figure 1-2 Example of Machine Language Instruction

Note from the example above that the operation code is "FA". The first operand as illustrated in Figure 1-2 consists of the lengths which determine the size of the numbers to be added and the address of the first number to be added. The second operand contains the main storage location of the second number to be used in the add operation.

The instruction illustrated in Figure 1-2 is an example of a "machine-language" instruction, that is, an instruction in a format which the controlling circuitry of the computer can understand and which will cause the computer to perform the desired operation. In the early 1950's, all programming was done in machine language. Although machine language is not a complex language, it is difficult for programmers to work with. They found writing programs in the language awkard, inconvenient, and subject to many errors. Because machine language was difficult to work with, "symbolic programming languages" were developed. These programming languages used symbols to represent machine language instructions. An example of a symbolic programming language is the "Assembler Language" which may be used on the System/360 computer. The assembler language provides for a series of symbolic representations of machine operation codes. These symbolic representations are often called "Mnemonic operation codes".

The following chart illustrates several of the instructions which are available on the System/360 and the corresponding Mnemonic and Machine Language instructions.

Instruction	Mnemonic Operation Code	Machine Operation Code
Add Decimal	AP	FA
Compare Decimal	CP	F9
Divide Decimal	DP	FD
Edit	ED	DE
Edit and Mark	EDMK	DF
Multiply Decimal	MP	FC
Subtract Decimal	SP	FB
Zero and Add	ZAP	F8

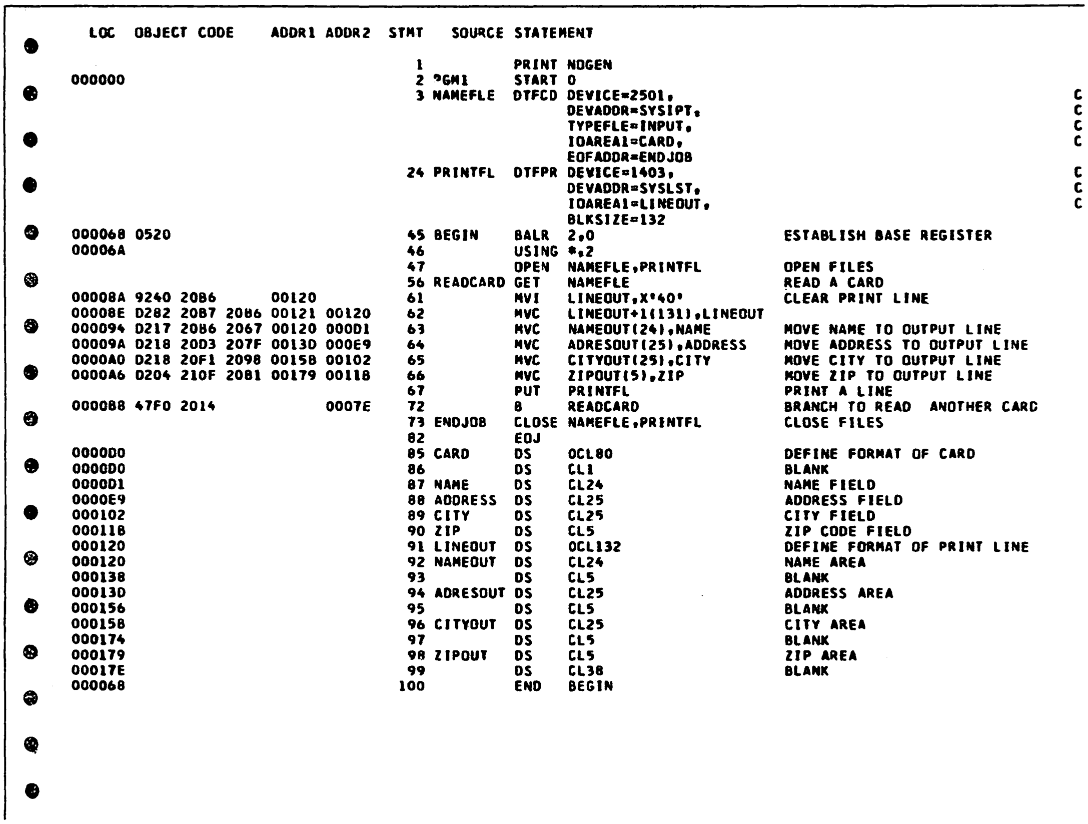
Figure 1-3 Mnemonic and Machine Language Operation Codes

Note from the chart above that the Mnemonic Operation Codes correspond somewhat to the function of the instruction. For example, the Mnemonic Operation code for the add instruction is AP. The corresponding Machine Language code is FA.

In order to have the computer solve a problem, the user must give it a set of step-by-step instructions that will result in the solution of the given problem. Such a set of instructions is called a "program". Guided by a program stored in its memory, a computer can solve commercial and scientific problems at very rapid speeds and with remarkable accuracy.

As noted, symbolic programming languages were developed in order to ease the task of programming. An example of a program written in System/360 Assembler Language is illustrated below.

EXAMPLE



```
  LOC  OBJECT CODE    ADDR1 ADDR2 STMT   SOURCE STATEMENT

                                     1          PRINT NOGEN
000000                               2 PGM1     START 0
                                     3 NAMEFLE  DTFCD DEVICE=2501,                                         C
                                                      DEVADDR=SYSIPT,                                      C
                                                      TYPEFLE=INPUT,                                       C
                                                      IOAREA1=CARD,                                        C
                                                      EOFADDR=ENDJOB
                                    24 PRINTFL  DTFPR DEVICE=1403,                                         C
                                                      DEVADDR=SYSLST,                                      C
                                                      IOAREA1=LINEOUT,                                     C
                                                      BLKSIZE=132
000068 0520                         45 BEGIN    BALR  2,0                    ESTABLISH BASE REGISTER
00006A                              46          USING *,2
                                    47          OPEN  NAMEFLE,PRINTFL        OPEN FILES
                                    56 READCARD GET   NAMEFLE                READ A CARD
00008A 9240 20B6          00120     61          MVI   LINEOUT,X'40'          CLEAR PRINT LINE
00008E D282 20B7 2086 00121 00120   62          MVC   LINEOUT+1(131),LINEOUT
000094 D217 20B6 2067 00120 000D1   63          MVC   NAMEOUT(24),NAME       MOVE NAME TO OUTPUT LINE
00009A D218 20D3 207F 0013D 000E9   64          MVC   ADRESOUT(25),ADDRESS   MOVE ADDRESS TO OUTPUT LINE
0000A0 D218 20F1 2098 0015B 00102   65          MVC   CITYOUT(25),CITY       MOVE CITY TO OUTPUT LINE
0000A6 D204 210F 20B1 00179 0011B   66          MVC   ZIPOUT(5),ZIP          MOVE ZIP TO OUTPUT LINE
                                    67          PUT   PRINTFL                PRINT A LINE
0000B8 47F0 2014            0007E   72          B     READCARD               BRANCH TO READ  ANOTHER CARD
                                    73 ENDJOB   CLOSE NAMEFLE,PRINTFL        CLOSE FILES
                                    82          EOJ
0000D0                              85 CARD     DS    0CL80                  DEFINE FORMAT OF CARD
0000D0                              86          DS    CL1                    BLANK
0000D1                              87 NAME     DS    CL24                   NAME FIELD
0000E9                              88 ADDRESS  DS    CL25                   ADDRESS FIELD
000102                              89 CITY     DS    CL25                   CITY FIELD
00011B                              90 ZIP      DS    CL5                    ZIP CODE FIELD
000120                              91 LINEOUT  DS    0CL132                 DEFINE FORMAT OF PRINT LINE
000120                              92 NAMEOUT  DS    CL24                   NAME AREA
000138                              93          DS    CL5                    BLANK
00013D                              94 ADRESOUT DS    CL25                   ADDRESS AREA
000156                              95          DS    CL5                    BLANK
00015B                              96 CITYOUT  DS    CL25                   CITY AREA
000174                              97          DS    CL5                    BLANK
000179                              98 ZIPOUT   DS    CL5                    ZIP AREA
00017E                              99          DS    CL38                   BLANK
000068                             100          END   BEGIN
```

Figure 1-4 Example of System/360 Assembler Language Program

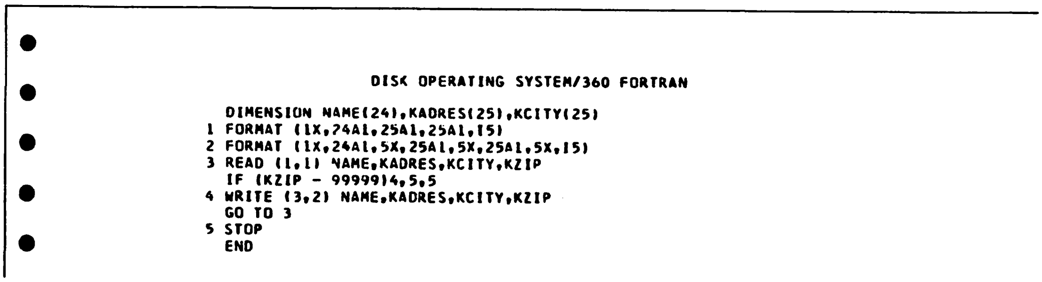
From the example above it can be seen that the program consists of a series of machine oriented instructions which cause the computer to process the data.

FORTRAN

Although the use of symbolic programming languages was far superior to the use of machine language programs, it was also recognized that in order to effectively use a computer system as a tool for the solution of scientific and business problems, there was a need for "high-level" programming languages, that is, programming languages which were not machine-oriented.

In 1957, International Business Machines released the specifications for a high-level scientific oriented language called FORTRAN (FORmula TRANslation). The programming language substantially relieved the user from having a detailed knowledge of the internal operation of an electronic data processing system, and allowed the programmer to define the input to be processed, the calculations to be performed, and the output to be produced in a format far removed from the internal operation of the computer. With FORTRAN, the programmer was not required to have a detailed knowledge of internal data representation, storage locations, etc. An example of a FORTRAN program is illustrated below.

EXAMPLE



```
                         DISK OPERATING SYSTEM/360 FORTRAN

  DIMENSION NAME(24),KADRES(25),KCITY(25)
1 FORMAT (1X,24A1,25A1,25A1,I5)
2 FORMAT (1X,24A1,5X,25A1,5X,25A1,5X,I5)
3 READ (1,1) NAME,KADRES,KCITY,KZIP
  IF (KZIP - 99999)4,5,5
4 WRITE (3,2) NAME,KADRES,KCITY,KZIP
  GO TO 3
5 STOP
  END
```

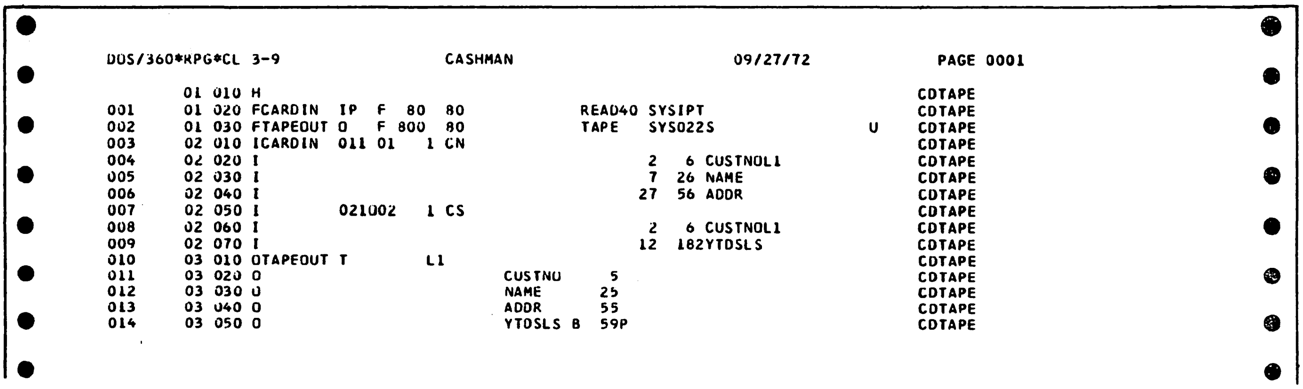
Figure 1-5 Example of FORTRAN Program

In the example above it can be seen that the format of the language is much closer to mathematical notation than it is to the machine-oriented language, Assembler Language. It is this way so that the scientific or mathematical programmer is able to express the processing to be performed in a manner similar to the notation he is familiar with.

RPG

RPG is another high-level language designed specifically for report writing and file maintenance applications. The language facilitates producing programs for a wide variety of reports ranging from a simple listing to a complete report that incorporates calculations and editing. A source listing of an RPG program is illustrated in Figure 1-6.

EXAMPLE



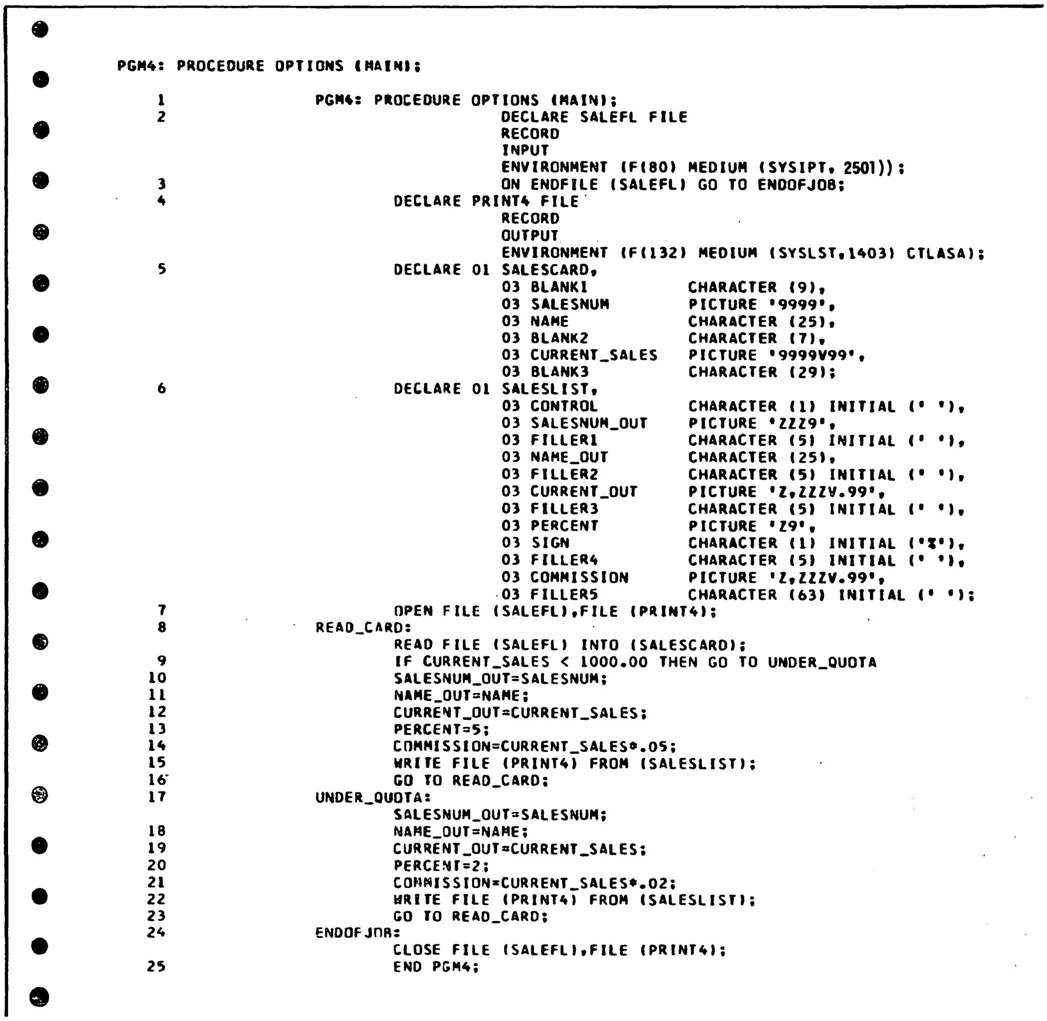
```
DOS/360*RPG*CL 3-9                CASHMAN                          09/27/72              PAGE 0001

          01 010 H                                                                     CDTAPE
001       01 020 FCARDIN  IP  F  80  80                READ40 SYSIPT                   CDTAPE
002       01 030 FTAPEOUT O   F 800  80                TAPE   SYS022S             U    CDTAPE
003       02 010 ICARDIN  011 01    1 CN                                               CDTAPE
004       02 020 I                                          2   6 CUSTNOL1             CDTAPE
005       02 030 I                                          7  26 NAME                 CDTAPE
006       02 040 I                                         27  56 ADDR                 CDTAPE
007       02 050 I        021002    1 CS                                               CDTAPE
008       02 060 I                                          2   6 CUSTNOL1             CDTAPE
009       02 070 I                                         12  182YTDSLS               CDTAPE
010       03 010 OTAPEOUT T        L1                                                  CDTAPE
011       03 020 O                          CUSTNO     5                               CDTAPE
012       03 030 O                          NAME      25                               CDTAPE
013       03 040 O                          ADDR      55                               CDTAPE
014       03 050 O                          YTDSLS B  59P                              CDTAPE
```

Figure 1-6 RPG Program

PL/I

PL/I provides the programmer with a unified problem-oriented language for efficiently programming either scientific or commercial problems, as well as problems that can best be solved with a combination of scientific and commercial computing techniques. The example in Figure 1-7 is a source listing of a PL/I program.

EXAMPLE



```
PGM4: PROCEDURE OPTIONS (MAIN);

   1             PGM4: PROCEDURE OPTIONS (MAIN);
   2                              DECLARE SALEFL FILE
                                  RECORD
                                  INPUT
                                  ENVIRONMENT (F(80) MEDIUM (SYSIPT, 2501));
   3                              ON ENDFILE (SALEFL) GO TO ENDOFJOB;
   4                      DECLARE PRINT4 FILE
                                  RECORD
                                  OUTPUT
                                  ENVIRONMENT (F(132) MEDIUM (SYSLST,1403) CTLASA);
   5                      DECLARE 01 SALESCARD,
                                  03 BLANK1           CHARACTER (9),
                                  03 SALESNUM         PICTURE '9999',
                                  03 NAME             CHARACTER (25),
                                  03 BLANK2           CHARACTER (7),
                                  03 CURRENT_SALES    PICTURE '9999V99',
                                  03 BLANK3           CHARACTER (29);
   6                      DECLARE 01 SALESLIST,
                                  03 CONTROL          CHARACTER (1) INITIAL (' '),
                                  03 SALESNUM_OUT     PICTURE 'ZZZ9',
                                  03 FILLER1          CHARACTER (5) INITIAL (' '),
                                  03 NAME_OUT         CHARACTER (25),
                                  03 FILLER2          CHARACTER (5) INITIAL (' '),
                                  03 CURRENT_OUT      PICTURE 'Z,ZZZV.99',
                                  03 FILLER3          CHARACTER (5) INITIAL (' '),
                                  03 PERCENT          PICTURE 'Z9',
                                  03 SIGN             CHARACTER (1) INITIAL ('%'),
                                  03 FILLER4          CHARACTER (5) INITIAL (' '),
                                  03 COMMISSION       PICTURE 'Z,ZZZV.99',
                                  03 FILLER5          CHARACTER (63) INITIAL (' ');
   7                      OPEN FILE (SALEFL),FILE (PRINT4);
   8             READ_CARD:
                          READ FILE (SALEFL) INTO (SALESCARD);
   9                      IF CURRENT_SALES < 1000.00 THEN GO TO UNDER_QUOTA
  10                      SALESNUM_OUT=SALESNUM;
  11                      NAME_OUT=NAME;
  12                      CURRENT_OUT=CURRENT_SALES;
  13                      PERCENT=5;
  14                      COMMISSION=CURRENT_SALES*.05;
  15                      WRITE FILE (PRINT4) FROM (SALESLIST);
  16                      GO TO READ_CARD;
  17             UNDER_QUOTA:
                          SALESNUM_OUT=SALESNUM;
  18                      NAME_OUT=NAME;
  19                      CURRENT_OUT=CURRENT_SALES;
  20                      PERCENT=2;
  21                      COMMISSION=CURRENT_SALES*.02;
  22                      WRITE FILE (PRINT4) FROM (SALESLIST);
  23                      GO TO READ_CARD;
  24             ENDOFJOB:
                          CLOSE FILE (SALEFL),FILE (PRINT4);
  25                      END PGM4;
```

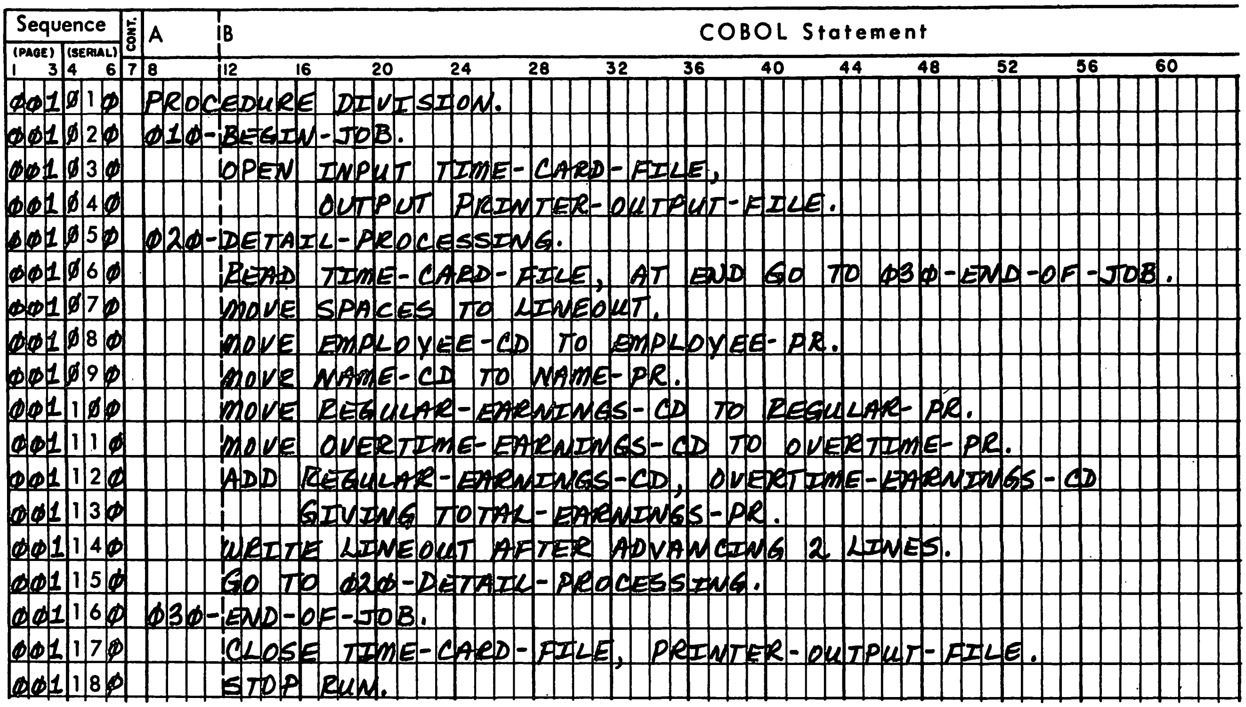
Figure 1-7 PL/I Program

COBOL

Soon after symbolic programming languages were introduced, it was recognized that there was also a need for a "high-level" business oriented language. On May 28 of 1959 a meeting was called by the Department of Defense, at the suggestion of a small group of computer users, manufacturers, and universities to discuss the problem of developing a common business language for computers. This original committee meeting was called CODASYL (Conference on DAta SYstems Languages). The purpose of this conference was to consider both the desirability and feasibility of establishing a common language for use by all manufacturers of electronic data processing systems. There was almost unanimous agreement that the project was both desirable and feasible. A short range committee was established and composed of representatives from six manufacturers (Burroughs, IBM, Honeywell, RCA, Remington Rand, and Sylvania Electric Products); two government agencies, the Air Material Command, and Department of the Navy; and, in addition, a representative from the Bureau of Standards. The first specifications for this new programming language called COBOL (COmmon Business Oriented Language) were released in April of 1960. Since that time a number of revisions and additions have been made to the original language specifications. It is interesting to note that this was the first attempt to develop a common programming language in which there was cooperation of a group of competing manufacturers of computer systems. Thus, COBOL is the only language developed through the cooperative efforts of various manufacturers within the data processing industry. Today, COBOL, is one of the dominate programming languages used for the solution of business problems.

The purpose of COBOL was to provide a common business-oriented language. The word common was interpreted to mean that the programming language was to be compatible among a significant group of manufacturers. In addition, a basic objective was that the language was to be natural, where natural was defined as "English-like". COBOL was not designed to permit concise writing. On the contrary, the language is very "wordy"; however, the benefit gained from this is the increased readability of the programs and the "built-in" documentation because of the readability of the language.

The following is an illustration of a COBOL program. The Procedure Division of a COBOL program describes the logic required in the solution of a problem. Notice that the program is readable without having a proficiency in the structure of the language.



```
001010 PROCEDURE DIVISION.
001020 010-BEGIN-JOB.
001030     OPEN INPUT TIME-CARD-FILE,
001040         OUTPUT PRINTER-OUTPUT-FILE.
001050 020-DETAIL-PROCESSING.
001060     READ TIME-CARD-FILE, AT END GO TO 030-END-OF-JOB.
001070     MOVE SPACES TO LINEOUT.
001080     MOVE EMPLOYEE-CD TO EMPLOYEE-PR.
001090     MOVE NAME-CD TO NAME-PR.
001100     MOVE REGULAR-EARNINGS-CD TO REGULAR-PR.
001110     MOVE OVERTIME-EARNINGS-CD TO OVERTIME-PR.
001120     ADD REGULAR-EARNINGS-CD, OVERTIME-EARNINGS-CD
001130         GIVING TOTAL-EARNINGS-PR.
001140     WRITE LINEOUT AFTER ADVANCING 2 LINES.
001150     GO TO 020-DETAIL-PROCESSING.
001160 030-END-OF-JOB.
001170     CLOSE TIME-CARD-FILE, PRINTER-OUTPUT-FILE.
001180     STOP RUN.
```

Figure 1-8 Example of COBOL Program

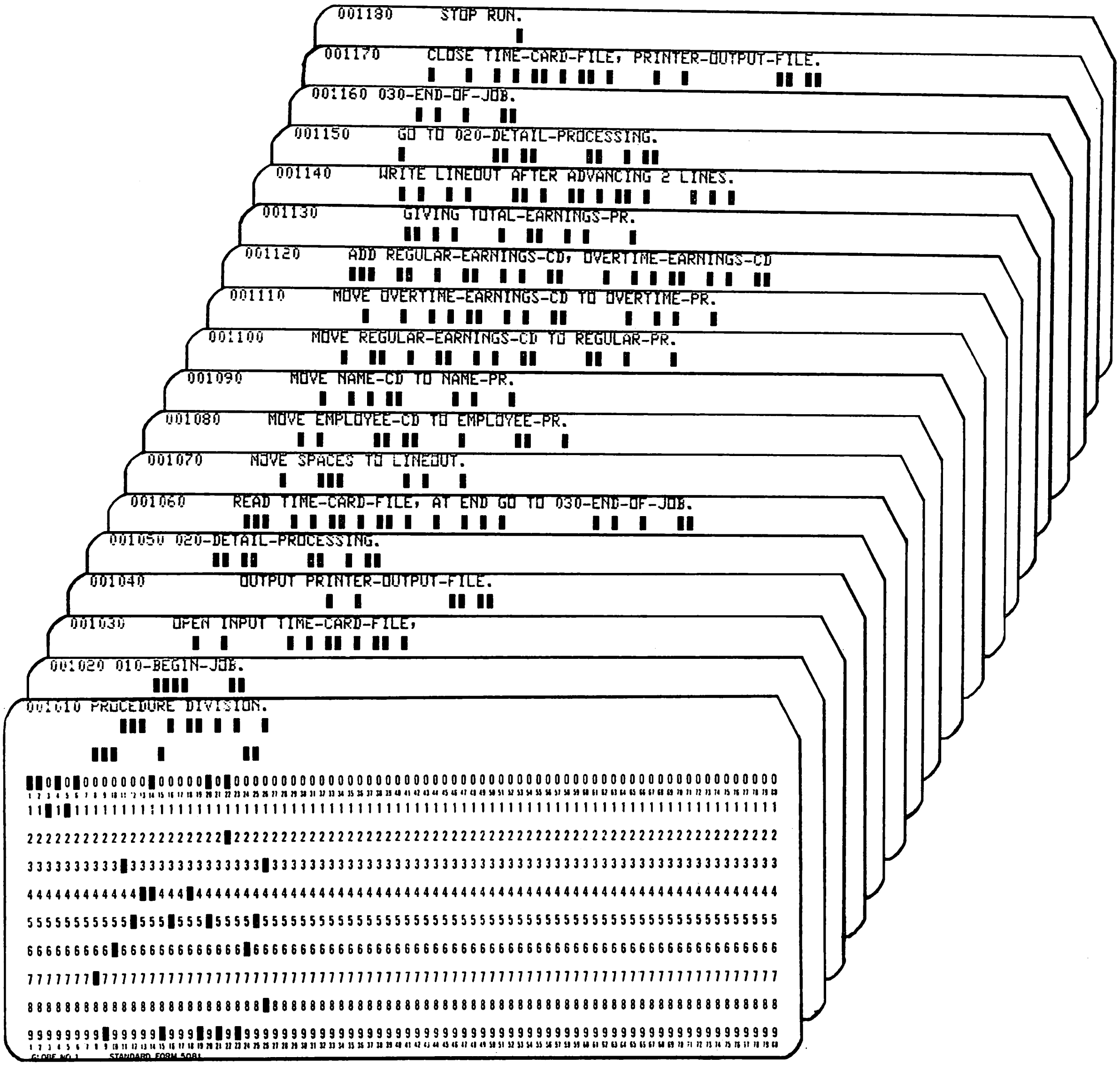
ANSI COBOL

With the advent of the COBOL programming language, many different versions of the language evolved from different manufacturers and it was soon recognized that it would be advantageous to have a "standard" COBOL which was uniform on all machines made by all manufacturers. Toward that end, the American Standards Institute, which had established committees dealing with all aspects of computers and information processing, formed a special committee which was given the responsibility for developing COBOL Standards.

The first meeting of the committee was held in New York City in January, 1963. The principal work of developing COBOL Standards was achieved by four special subcommittees, and in 1968, the American National Standard Institute (ANSI) COBOL was released. The use of ANSI COBOL virtually assures machine-independence as all ANSI COBOL compilers will contain the same basic components and specifications. Since the release of ANSI COBOL, some manufacturers have added extensions to the language, but the basic standard COBOL is found on almost all machines available for business data processing.

PROCESSING OF COBOL PROGRAMS

For processing on the computer, the COBOL program is punched onto cards for reading by an input device. The following diagram illustrates a segment of a COBOL program as it appears on punched cards. Note that for each line of coding on the form (Figure 1-8), a single punched card is required.



```
001180     STOP RUN.
001170     CLOSE TIME-CARD-FILE, PRINTER-OUTPUT-FILE.
001160 030-END-OF-JOB.
001150     GO TO 020-DETAIL-PROCESSING.
001140     WRITE LINEOUT AFTER ADVANCING 2 LINES.
001130         GIVING TOTAL-EARNINGS-PR.
001120     ADD REGULAR-EARNINGS-CD, OVERTIME-EARNINGS-CD
001110     MOVE OVERTIME-EARNINGS-CD TO OVERTIME-PR.
001100     MOVE REGULAR-EARNINGS-CD TO REGULAR-PR.
001090     MOVE NAME-CD TO NAME-PR.
001080     MOVE EMPLOYEE-CD TO EMPLOYEE-PR.
001070     MOVE SPACES TO LINEOUT.
001060     READ TIME-CARD-FILE, AT END GO TO 030-END-OF-JOB.
001050 020-DETAIL-PROCESSING.
001040         OUTPUT PRINTER-OUTPUT-FILE.
001030     OPEN INPUT TIME-CARD-FILE,
001020 010-BEGIN-JOB.
001010 PROCEDURE DIVISION.
```

Figure 1-9 Segment of COBOL Program Punched on Cards

It should be noted that the computer can only understand instructions on a machine language level; therefore, COBOL sentences written by the programmer must be converted to machine language instructions prior to processing data. For example, the COBOL word "ADD" must be converted to the symbols FA for the computer to "understand" the function to be performed.

Because computers only understand instructions in machine language form, symbolic "translators" were developed. The "translator" program that enables a computer to translate COBOL statements into its own machine language is called a "compiler". A programming language and its associated "compiler" is called a "programming system". The programming languages used by an electronic data processing system are frequently called the computer's SOFTWARE.

The following diagram illustrates the basic steps to compile a COBOL Language program.

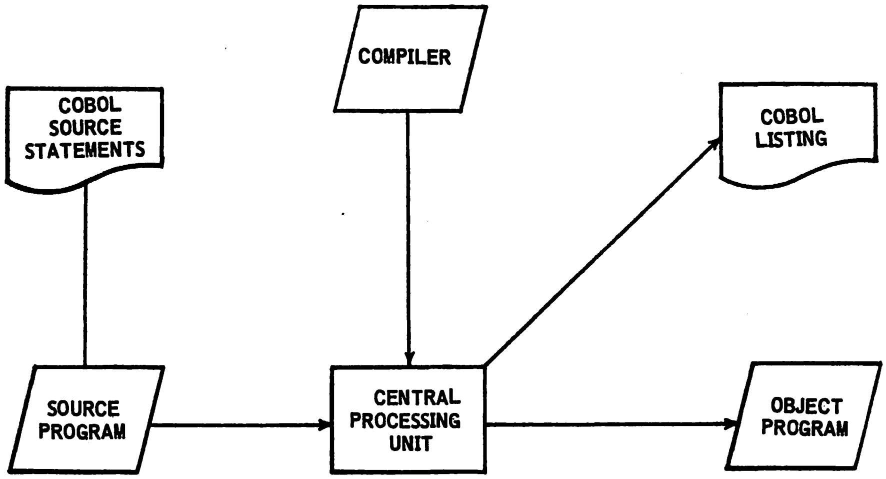


Figure 1-10 Steps in Compiling a Program

To "convert" a program from a symbolic form to machine language, the Compiler is read into the storage unit of the computer followed by the "Source COBOL Program". The compiler then "translates" the symbolic statements of the source COBOL program into a machine language form and punches an OBJECT PROGRAM if desired. The Object Program is the machine language instructions generated by the Compiler from the COBOL Program.

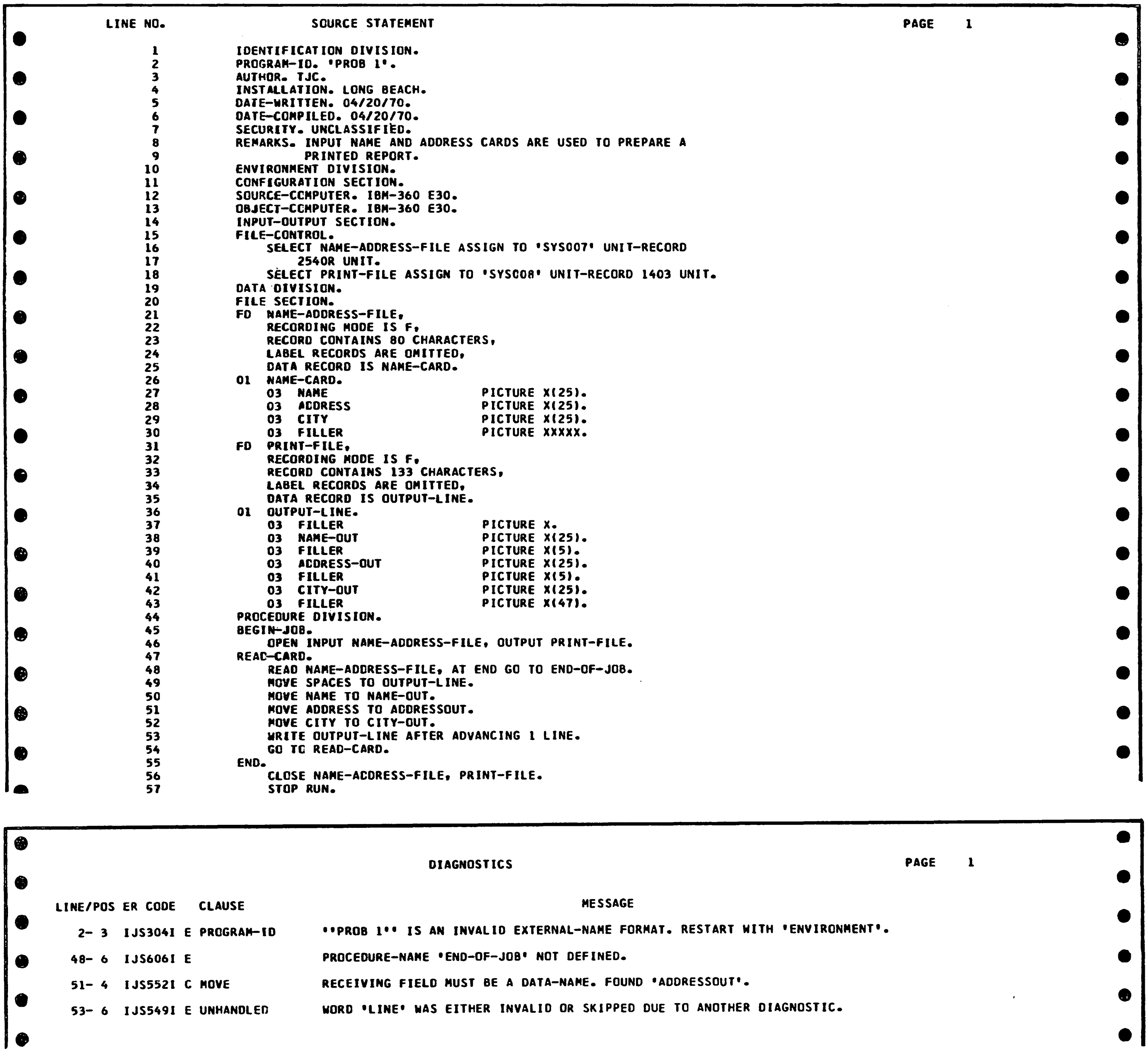
The "compiler" also provides auxilary functions that assist the programmer in checking and documenting programs. Some of these functions are:

Program Listings: A listing of the source program statements may be produced by the COBOL compiler for each source program compiled.

Error Indications: As a source program is compiled, it is analyzed for actual or potential errors in the use of the COBOL language. Detected errors are indicated in the program listing.

The following is a segment of a listing of a COBOL program generated by the computer and "diagnostics" or errors detected by the compiler. These errors must be corrected before the computer can execute the object program and process the data.

EXAMPLE



```
        LINE NO.                    SOURCE STATEMENT                                                     PAGE    1

            1          IDENTIFICATION DIVISION.
            2          PROGRAM-ID. 'PROB 1'.
            3          AUTHOR. TJC.
            4          INSTALLATION. LONG BEACH.
            5          DATE-WRITTEN. 04/20/70.
            6          DATE-COMPILED. 04/20/70.
            7          SECURITY. UNCLASSIFIED.
            8          REMARKS. INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A
            9                   PRINTED REPORT.
           10          ENVIRONMENT DIVISION.
           11          CONFIGURATION SECTION.
           12          SOURCE-CCMPUTER. IBM-360 E30.
           13          OBJECT-CCMPUTER. IBM-360 E30.
           14          INPUT-OUTPUT SECTION.
           15          FILE-CONTROL.
           16              SELECT NAME-ADDRESS-FILE ASSIGN TO 'SYS007' UNIT-RECORD
           17                  2540R UNIT.
           18              SELECT PRINT-FILE ASSIGN TO 'SYSC08' UNIT-RECORD 1403 UNIT.
           19          DATA DIVISION.
           20          FILE SECTION.
           21          FD  NAME-ADDRESS-FILE,
           22              RECORDING MODE IS F,
           23              RECORD CONTAINS 80 CHARACTERS,
           24              LABEL RECORDS ARE OMITTED,
           25              DATA RECORD IS NAME-CARD.
           26          01  NAME-CARD.
           27              03  NAME                        PICTURE X(25).
           28              03  ACDRESS                     PICTURE X(25).
           29              03  CITY                        PICTURE X(25).
           30              03  FILLER                      PICTURE XXXXX.
           31          FD  PRINT-FILE,
           32              RECORDING MODE IS F,
           33              RECORD CONTAINS 133 CHARACTERS,
           34              LABEL RECORDS ARE OMITTED,
           35              DATA RECORD IS OUTPUT-LINE.
           36          01  OUTPUT-LINE.
           37              03  FILLER                      PICTURE X.
           38              03  NAME-OUT                    PICTURE X(25).
           39              03  FILLER                      PICTURE X(5).
           40              03  ACDRESS-OUT                 PICTURE X(25).
           41              03  FILLER                      PICTURE X(5).
           42              03  CITY-OUT                    PICTURE X(25).
           43              03  FILLER                      PICTURE X(47).
           44          PROCEDURE DIVISION.
           45          BEGIN-JOB.
           46              OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.
           47          REAC-CARD.
           48              READ NAME-ADDRESS-FILE, AT END GO TO END-OF-JOB.
           49              MOVE SPACES TO OUTPUT-LINE.
           50              MOVE NAME TO NAME-OUT.
           51              MOVE ADDRESS TO ACDRESSOUT.
           52              MOVE CITY TO CITY-OUT.
           53              WRITE OUTPUT-LINE AFTER ADVANCING 1 LINE.
           54              GO TC READ-CARD.
           55          END.
           56              CLOSE NAME-ACDRESS-FILE, PRINT-FILE.
           57              STOP RUN.
```

```
                                           DIAGNOSTICS                                               PAGE    1

LINE/POS ER CODE   CLAUSE                                          MESSAGE

   2- 3  IJS304I E PROGRAM-ID      ''PROB 1'' IS AN INVALID EXTERNAL-NAME FORMAT. RESTART WITH 'ENVIRONMENT'.

  48- 6  IJS606I E                 PROCEDURE-NAME 'END-OF-JOB' NOT DEFINED.

  51- 4  IJS552I C MOVE            RECEIVING FIELD MUST BE A DATA-NAME. FOUND 'ADDRESSOUT'.

  53- 6  IJS549I E UNHANDLED       WORD 'LINE' WAS EITHER INVALID OR SKIPPED DUE TO ANOTHER DIAGNOSTIC.
```

Figure 1-11 Example of COBOL Program and Diagnostics

SUMMARY

Even a brief introduction to COBOL should make it apparent that COBOL is oriented to the functions of problem solving. The programmer need devote little attention to the internal characteristics of the computer being used or the internal operation of the machine. A programming language which possesses such characteristics is called a "high-level" or "problem-oriented" language.

COBOL – Advantages

Using COBOL to code source programs has some decided advantages when compared to "machine-oriented" programming languages. They are:

1. The programmer's attention is concentrated on the problem rather than on the hardware characteristics of the machine used. Thus, a detailed knowledge of the object language instruction set, relative addressing and other machine-oriented subjects is not normally needed by the COBOL programmer.

2. COBOL is comparatively easy to learn. Because the notation is problem oriented, the actual program is generally easier to write. An additional advantage is that once written, the program is to a great extent self-documenting and thus, easy to maintain.

3. A problem written in COBOL is generally easier to correct or "debug" than one written in a symbolic assembler type language. The diagnostics which the COBOL compiler performs on the source program provides the programmer with an automatic check of the accuracy of much of his coding.

4. An important consideration in the selection of a programming language is the ease of conversion from one computer to another. Because ANSI COBOL was designed as a standard language for all computers, it tends to be one of the most widely used programming languages on computers from all manufacturers.

5. An important advantage to a high-level programming language is that it usually reduces the total amount of programming time required in the solution of typical problems. Typically it is faster to program in a high-level language than to program in a machine-oriented language.

COBOL – Disadvantages

1. The time required to compile a COBOL program might be greater than with a machine-oriented programming language.

2. A disadvantage which may significantly effect production runs is the fact that high-level programming languages such as COBOL normally do not produce as efficient object code as when programming in a machine-oriented language.

3. It is sometimes necessary for a COBOL programmer to have a knowledge of machine language code, for if diagnostics do not produce a clue as to the cause of the failure of the program to process the data it may be necessary to look at a "core dump". Thus, the COBOL programmer may be required to have a proficiency in two programming languages.

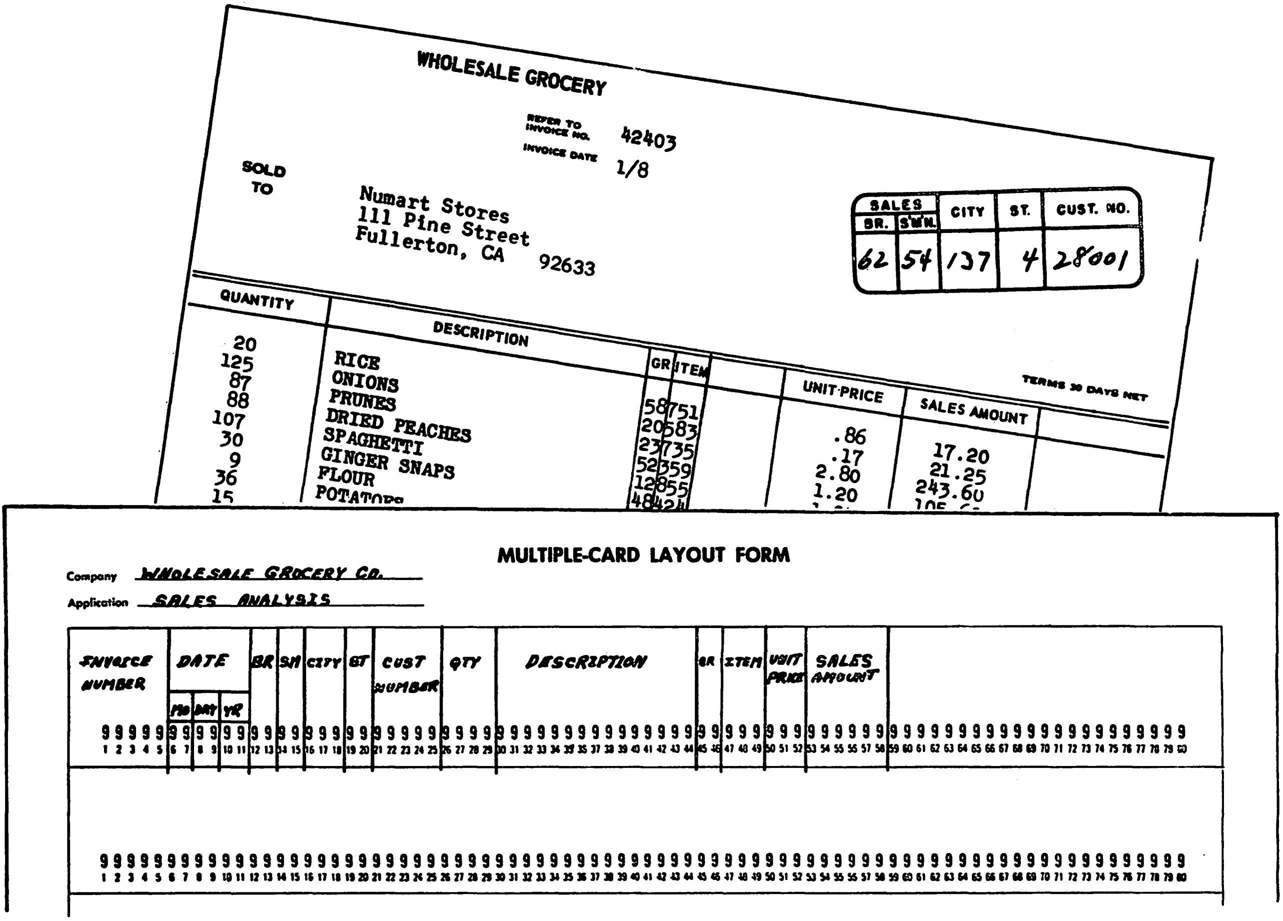
4. In some problems there are operations which cannot be performed when using high-level languages. For example, the ability to manipulate individual bits is often difficult if not impossible with high-level languages.

PROGRAMMING STEPS

Whenever a problem is to be solved on the computer, the programmer must write the program which will process the data on the computer. Contrary to popular belief, however, all of a programmer's time is not spent in "writing" the program, that is, coding the program in a programming language. A great deal of the programmer's time is spent in analyzing a problem, determining the solution to the problem, testing the program to ensure that it works properly, and documenting the program. The various duties which are performed by the programmer are explained below.

1. Problem Analysis – The normal sequence of an application which is to be implemented on a computer is for a systems analyst to design the overall system, which is composed of one or more programs. The systems analyst gathers the information which is necessary to comprise the system from the eventual user of the system, from management, and from past experience which would indicate the type of processing which should be included in the system. He then places this information in a form which may be communicated to the programmer. Typically, the manner of presentation will include record formats, that is, the format of the input and output records which are to be processed by the program, printer spacing charts which are used to illustrate the format of a printed report, and some type of written narrative which will describe in detail the processing which is to take place within the system and within each program included in the system. The example in Figure 1-12 illustrates a record layout form and a printer spacing chart for a "typical" application involving the preparation of a sales analysis report.

Card Input



WHOLESALE GROCERY

REFER TO INVOICE NO. 42403
INVOICE DATE 1/8

SOLD TO
Numart Stores
111 Pine Street
Fullerton, CA 92633

SALES BR.	SALES S'MN.	CITY	ST.	CUST. NO.
62	54	137	4	28001

TERMS 30 DAYS NET

QUANTITY	DESCRIPTION	GR	ITEM		UNIT PRICE	SALES AMOUNT	
20	RICE	58	751		.86	17.20	
125	ONIONS	20	583		.17	21.25	
87	PRUNES	23	735		2.80	243.60	
88	DRIED PEACHES	52	359		1.20		
107	SPAGHETTI	12	855				
30	GINGER SNAPS	48					
9	FLOUR						
36							
15							

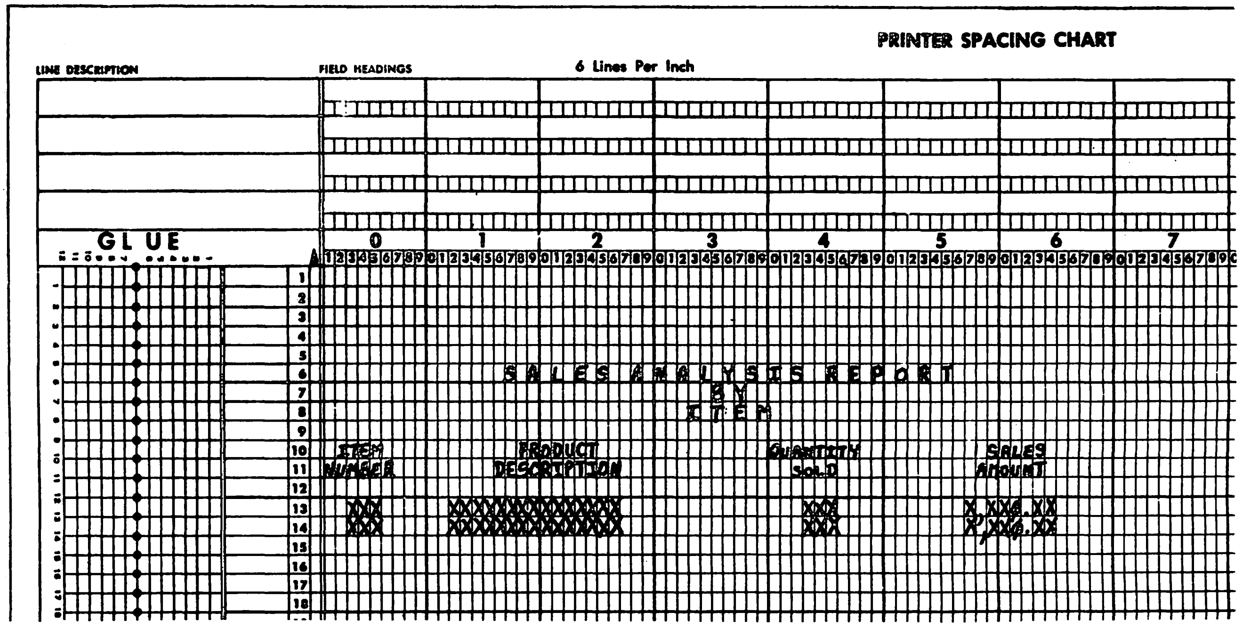
MULTIPLE-CARD LAYOUT FORM

Company WHOLESALE GROCERY CO.
Application SALES ANALYSIS

INVOICE NUMBER	DATE MO	DATE DAY	DATE YR	BR	SM	CITY	ST	CUST NUMBER	QTY	DESCRIPTION	GR	ITEM	UNIT PRICE	SALES AMOUNT	
1 2 3 4 5	6 7	8 9	10 11	12 13	14 15	16 17 18	19 20	21 22 23 24 25	26 27 28 29	30 31 32 33 34 35 36 37 38 39 40 41 42 43 44	45 46	47 48 49	50 51 52	53 54 55 56 57 58	59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

Sales Order and Multiple-Card Layout Form

Printer Spacing Chart



PRINTER SPACING CHART

LINE DESCRIPTION
FIELD HEADINGS
6 Lines Per Inch

GLUE

```
Line 6:                 S A L E S   A N A L Y S I S   R E P O R T
Line 7:                                B Y
Line 8:                              I T E M
Line 10:  ITEM           PRODUCT                  QUANTITY          SALES
Line 11: NUMBER        DESCRIPTION                  SOLD            AMOUNT
Line 13:  XXX        XXXXXXXXXXXXXXX                 XXX            X,XX0.XX
Line 14:  XXX        XXXXXXXXXXXXXXX                 XXX            X,XX0.XX
```

Figure 1-12 Example of Printer Spacing Chart and Multiple-Card Layout Form

In the example in Figure 1-12 it can be seen that a Multiple-Card Layout form is used to specify the layout of the card which is to be processed and that the printer spacing chart is used to illustrate the format of the report which is to be produced by the program.

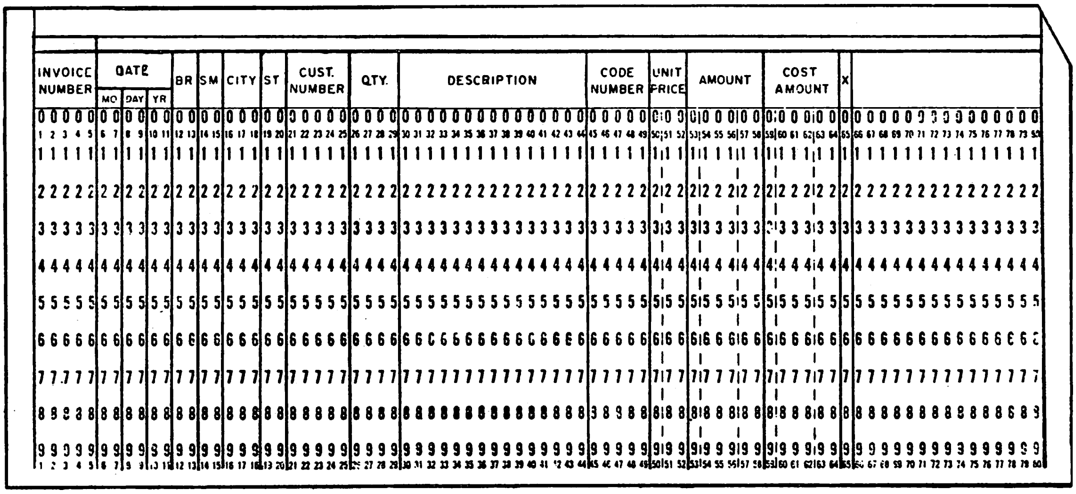
To layout a report on the Printer Spacing Chart, the programmer or analyst selects the print positions for the headings and the fields to be printed on the report and makes a notation in the selected positions. The numbers across the top of the printer spacing chart represent the actual print positions used by the computer printer. The numbers down the left are line numbers. There are six horizontal lines to an inch. Heading or constant informat to be printed is written on the printer spacing chart in the same form as it is to be printed on the report. Variable information is represented by placing an "X" in the appropriate position on the spacing chart. It should be noted that the position in a field where zero-suppression ends is indicated by a zero rather than an "X" and punctuation is shown as it would appear in edited amounts.

When the programmer writes the program to process the data to produce the report, the definition of the input and printed output will normally be in the form of a Multiple-Card Layout Form and a Printer Spacing Chart.

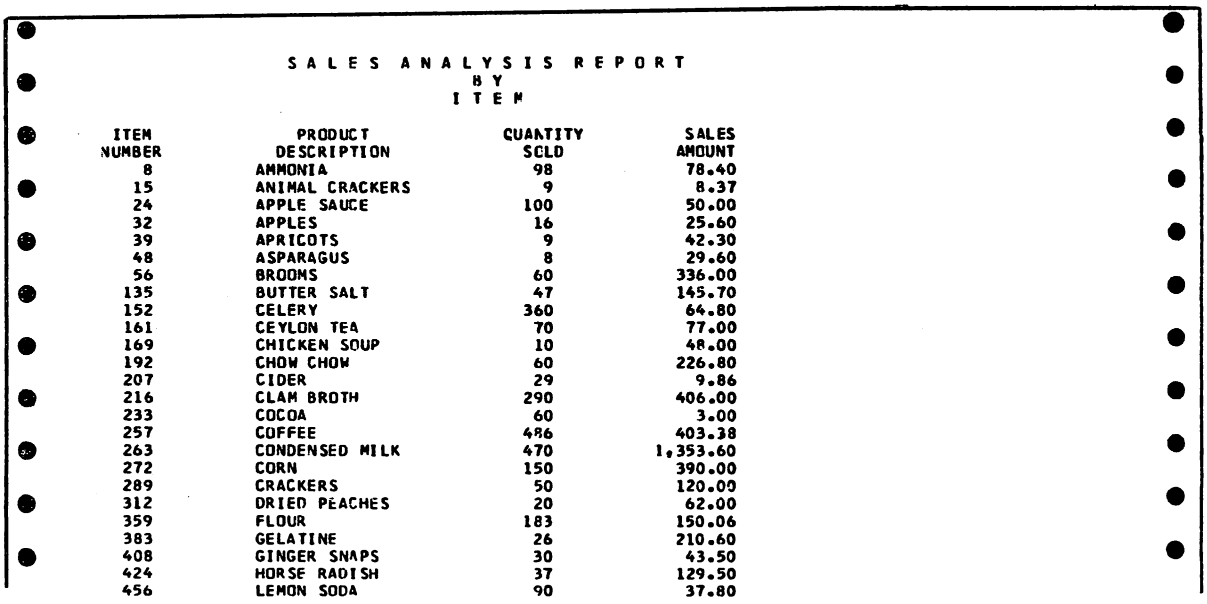
An example of the printed report and the actual card which would be used as a result of the layouts presented in Figure 1-12 are illustrated below.

EXAMPLE

Card Input



Printed Output



```
                        S A L E S   A N A L Y S I S   R E P O R T
                                          B Y
                                        I T E M

   ITEM                 PRODUCT                  CUANTITY          SALES
  NUMBER              DESCRIPTION                  SCLD            AMOUNT
     8              AMMONIA                         98              78.40
    15              ANIMAL CRACKERS                  9               8.37
    24              APPLE SAUCE                    100              50.00
    32              APPLES                          16              25.60
    39              APRICOTS                         9              42.30
    48              ASPARAGUS                        8              29.60
    56              BROOMS                          60             336.00
   135              BUTTER SALT                     47             145.70
   152              CELERY                         360              64.80
   161              CEYLON TEA                      70              77.00
   169              CHICKEN SOUP                    10              48.00
   192              CHOW CHOW                       60             226.80
   207              CIDER                           29               9.86
   216              CLAM BROTH                     290             406.00
   233              COCOA                           60               3.00
   257              COFFEE                         486             403.38
   263              CONDENSED MILK                 470           1,353.60
   272              CORN                           150             390.00
   289              CRACKERS                        50             120.00
   312              DRIED PEACHES                   20              62.00
   359              FLOUR                          183             150.06
   383              GELATINE                        26             210.60
   408              GINGER SNAPS                    30              43.50
   424              HORSE RADISH                    37             129.50
   456              LEMON SODA                      90              37.80
```

Figure 1-13 Example of Actual Print-Out and Card Input

As can be seen from the example above, the Sales report which is generated from the program is in the same format as the printer spacing chart in Figure 1-12. The same is true of the input card. Thus, through the use of the record layout forms, the analyst informs the programmer of the formats of the records to be processed by the program.

When the program specifications have been received from the systems analyst, the programmer must very carefully review the specifications so that every aspect of the program which he is to write is fully understood. Too much emphasis cannot be placed on the requirements that the programmer fully understand the processing which is to take place within a program. It is quite obvious that if the programmer does not understand what is required in a program, the program cannot possibly contain the proper coding to process the data. Beyond that basic truism, however, is the fact that a programmer can many times find errors or omissions within the system specification which must be corrected prior to the programming effort. An extremely detailed perusal of the system specifications must always be undertaken before any coding is performed on the program.

2. Determine Problem Solution – Once the programmer has determined that the problem to be solved is fully understood, he must then determine the method of solving the problem. This task normally involves determining the program logic which must be used in order to get the correct results in a program and is many times the most difficult step in solving a problem on the computer. The techniques which are used to determine the logical processing which is to take place within a program are many and varied. They include flowcharting, decision tables and narrative instructions. Whatever technique is used to graphically illustrate the logic required to process the data within the program, it is necessary that the logic include all of the processing which is to take place in enough detail that the programmer can write the source program from the logic illustrated.

3. Coding – After the logic which is to be incorporated into a program has been thoroughly understood, the program is coded to reflect the processing which is to occur. The programmer should take great card in preparing the source coding to include all of the steps which are required for the program. Attention to detail and accuracy in preparing the program will result in fewer errors when the program is tested and a more reliable program after it is in production.

4. Testing and Debugging: After the COBOL source program has been written, it is normally keypunched into cards so that it may be processed on the computer. After it has been keypunched, it is compiled on the computer as illustrated in Figure 1-10. As noted previously, when the compilation process is performed, errors are many times found in the coding due to improperly used statements or keypunching errors. These errors must be corrected by the programmer until the program is compiled with no errors in the source coding.

After a program has been successfully compiled, it must be tested in order to determine that the program is correct and the data is being processed properly. Testing a program involves several important steps which must be followed in order to ensure that a program is debugged completely and that it will operate successfully. These steps are explained below.

a). Desk Checking – After a program has been compiled, it should be "desk checked" prior to being executed on the computer. Desk checking refers to the process of the programmer "playing computer", that is, examining each source statement within the program as if the computer were processing the statements. By doing this, the programmer can find errors both in the use of statements and in the logic which will be performed by the program. Desk checking is a very important part of debugging a program because errors can be found which would otherwise not be found until the program were actually processed on the computer.

b). Preparing Test Data – When a program is to be tested, it should be tested with data which was specifically prepared to test the various routines within the program. It should not be tested with data which will be processed by the program after it has been put into production, that is, after it has been debugged and is ready to be run using actual "live" data. The reason for this is that "live" data, no matter how it is chosen, is not likely to contain all of the situations which may occur in the program. Thus, some occurrences which are programmed for will never be adequately tested. It is these routines which are probably going to fail when the program is in production. Instead, the programmer should design data which can be used to test all aspects of the program so that all routines and processing decisions are tested. The preparation of test data is a difficult and arduous task but a program cannot be adequately tested unless good test data is used.

c). Program Debugging – After the program has been desk checked and good test data has been prepared, the program may be tested on the computer. The number of test runs which will be required to completely debug a program is normally dependent upon the size and complexity of the program. Regardless of the size and complexity of the program, however, it is important that the program be completely debugged prior to being used in actual production runs. Too often it has been found that programs have not been entirely debugged prior to processing actual data and this leads to many problems both within the data processing department and with the users of the program. Testing a program is a difficult task and only when great care is used in testing will a properly debugged program result.

5. Documentation – Documentation is the process of recording the facts concerning a computer program. Included in the documentation will normally be a program narrative describing the routines and programming techniques used in the program, the source listing, the formats of the data which is processed by the program, and a sample running of the program including control cards which must be used and any reports which are produced by the program.

The documentation of a program is an often neglected job of the applications programmer. This neglect may be caused by data processing management, which desires the programmer to begin writing new programs, or by the programmer himself, who does not enjoy the rather mundane task of preparing narratives of his program and drawing file layouts, etc. It has been found, however, that proper documentation of a program is absolutely vital to the smooth functioning of a data processing department. In many applications, changes must be made to a program once it has been put into production and many times, the programmer making the changes is not the same person who originally wrote the program. Thus, if the "maintenance programmer", that is, the programmer who is making the changes, does not have sufficient documentation to indicate the processing which is occuring in the program and the methods used in the program, it becomes quite difficult to make changes which will work properly. It is, therefore, incumbent upon the programmer who originally writes a program to supply enough documentation to ensure that any programmer assigned to make changes to the program can do so with a minimum of time spent on the job of determining how the program processes the data.

SUMMARY

As was noted previously, the job of a programmer is much more diversified than just coding a program in a programming language. The jobs of testing a program and preparing documentation require a goodly portion of the programmer's time and effort. The tasks of problem analysis and problem solving also require much time and are quite critical to the solution of problems on the computer.

Regardless of the problem to be solved and the method of solution, however, the program must be written in order for the computer to process the data. As noted, a number of programming languages are available. The one to which the remainder of the text will be devoted, COBOL, is the major programming language in business applications and a knowledge of COBOL gives the programmer the knowledge to write business programs on most computers which are used in the business programming environment.

CHAPTER 2

INTRODUCTION TO COMPUTER PROGRAMMING

INPUT/OUTPUT OPERATIONS

INTRODUCTION TO COBOL

After the input for the solution of a problem has been established, the format of the printed report developed, and the logic of the program defined, the COBOL program can be written. Every COBOL program consists of four separate divisions with each division providing an essential and unique portion of the information required by the COBOL compiler in generating an object program. The four divisions of a COBOL program are:

1. IDENTIFICATION DIVISION – Identifies the program.
2. ENVIRONMENT DIVISION – Describes the computer used and the hardware units available.
3. DATA DIVISION – Describes the files, record layouts, and storage areas.
4. PROCEDURE DIVISION – States the program logic as expressed in the detail flowchart.

To illustrate the basic concepts of programming and the entries required to write a COBOL program, a sample problem is introduced in this chapter in which a file of Name and Address cards is used to prepare a Name and Address Listing. The input, output, flowchart, and coding are illustrated on the following pages.

Input

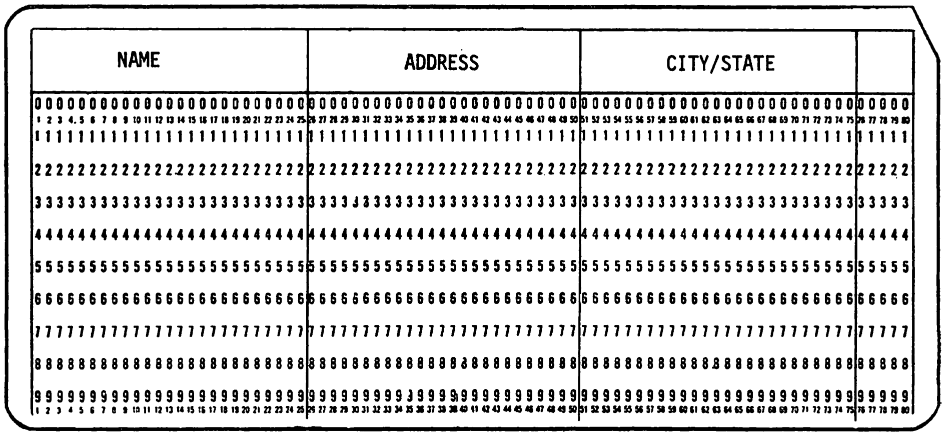
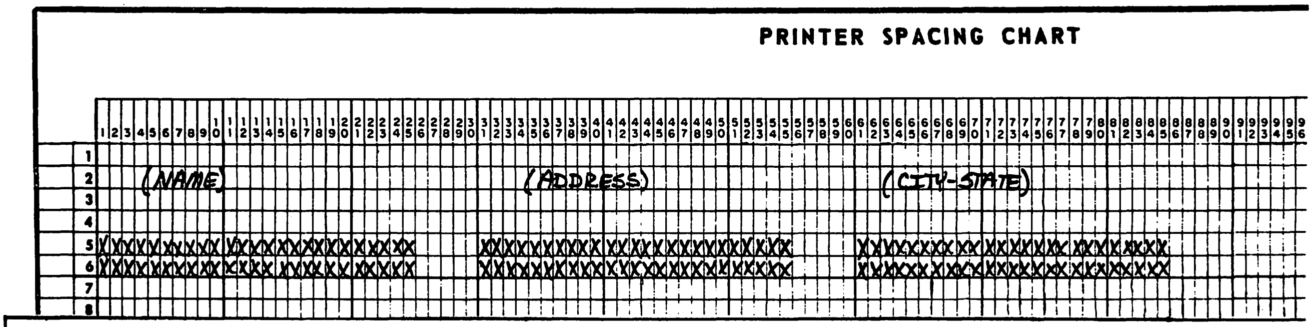


Figure 2-1 Name and Address Card

Output

Output for the sample problem is to consist of a printed report listing all fields on the cards. A printer spacing chart and the related output are illustrated below.



```
ARCHER ROBERT C.              113 PINE AVENUE               LONG BEACH, CALIFORNIA
ALLEN ROBERTA A.              3210 HENRY DRIVE              LAKEWOOD, CALIFORNIA
BATES ALLEN                   245 LOCUST STREET             LONG BEACH, CALIFORNIA
DAVIS WILLIAM G.              654 HENRI STREET              LAKEWOOD, CALIFORNIA
```

Figure 2-2 Printer Spacing Chart and Output

The basic computer system used to process the Name and Address File consists of a 2319 Disk Unit, a 2540 Card Reader, a 1403 Printer, and an IBM System/360 Model 30 with 65K bytes of main storage. It must be noted, however, that this program and the others contained within the text could be run on any computer which has an ANSI COBOL compiler.

These basic steps which are required to create the Name and Address Listing are illustrated below.

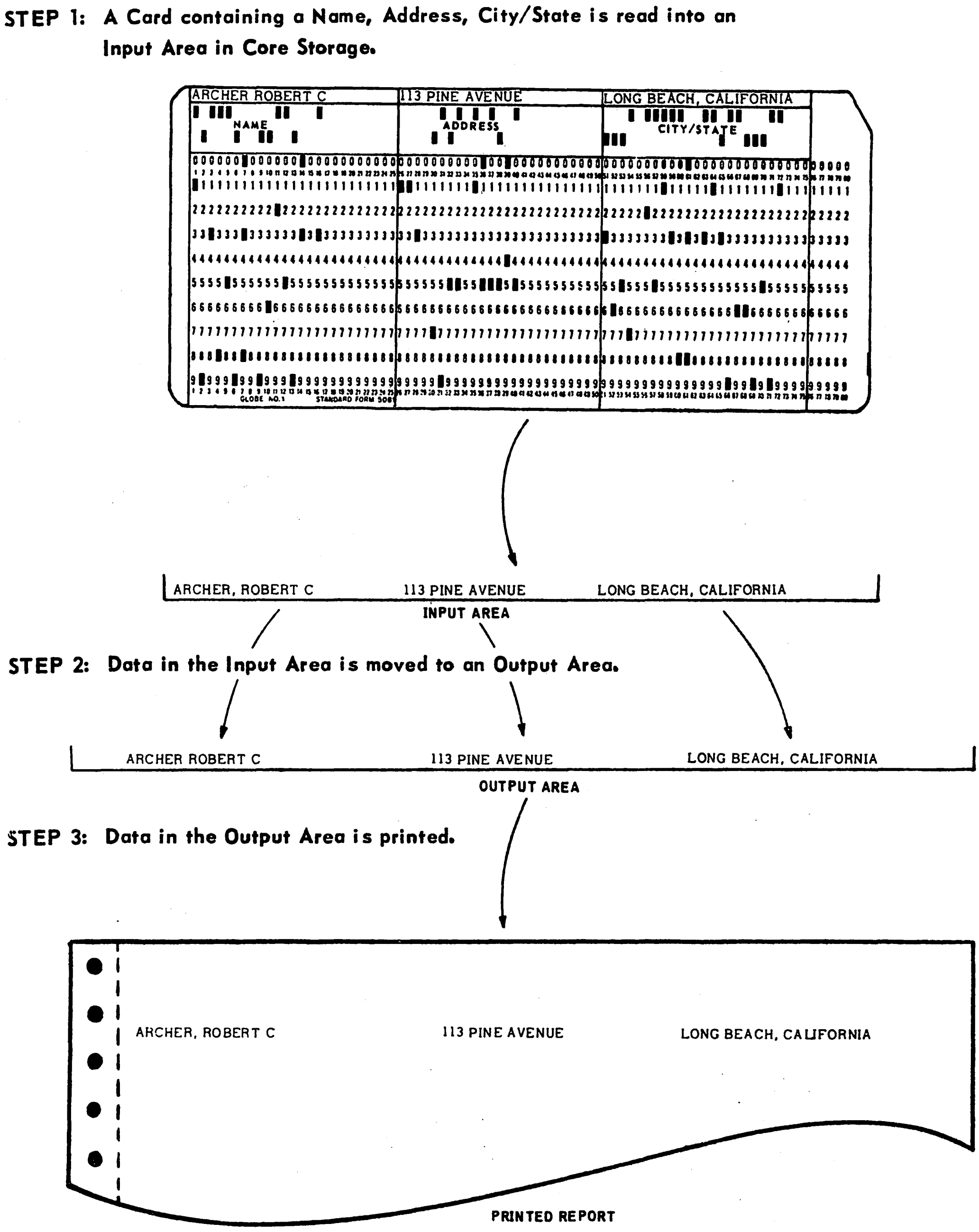

STEP 1: A Card containing a Name, Address, City/State is read into an Input Area in Core Storage.
ARCHER ROBERT C
113 PINE AVENUE
LONG BEACH, CALIFORNIA
NAME
ADDRESS
CITY/STATE
ARCHER, ROBERT C
113 PINE AVENUE
LONG BEACH, CALIFORNIA
INPUT AREA
STEP 2: Data in the Input Area is moved to an Output Area.
ARCHER ROBERT C
113 PINE AVENUE
LONG BEACH, CALIFORNIA
OUTPUT AREA
STEP 3: Data in the Output Area is printed.
ARCHER, ROBERT C
113 PINE AVENUE
LONG BEACH, CALIFORNIA
PRINTED REPORT

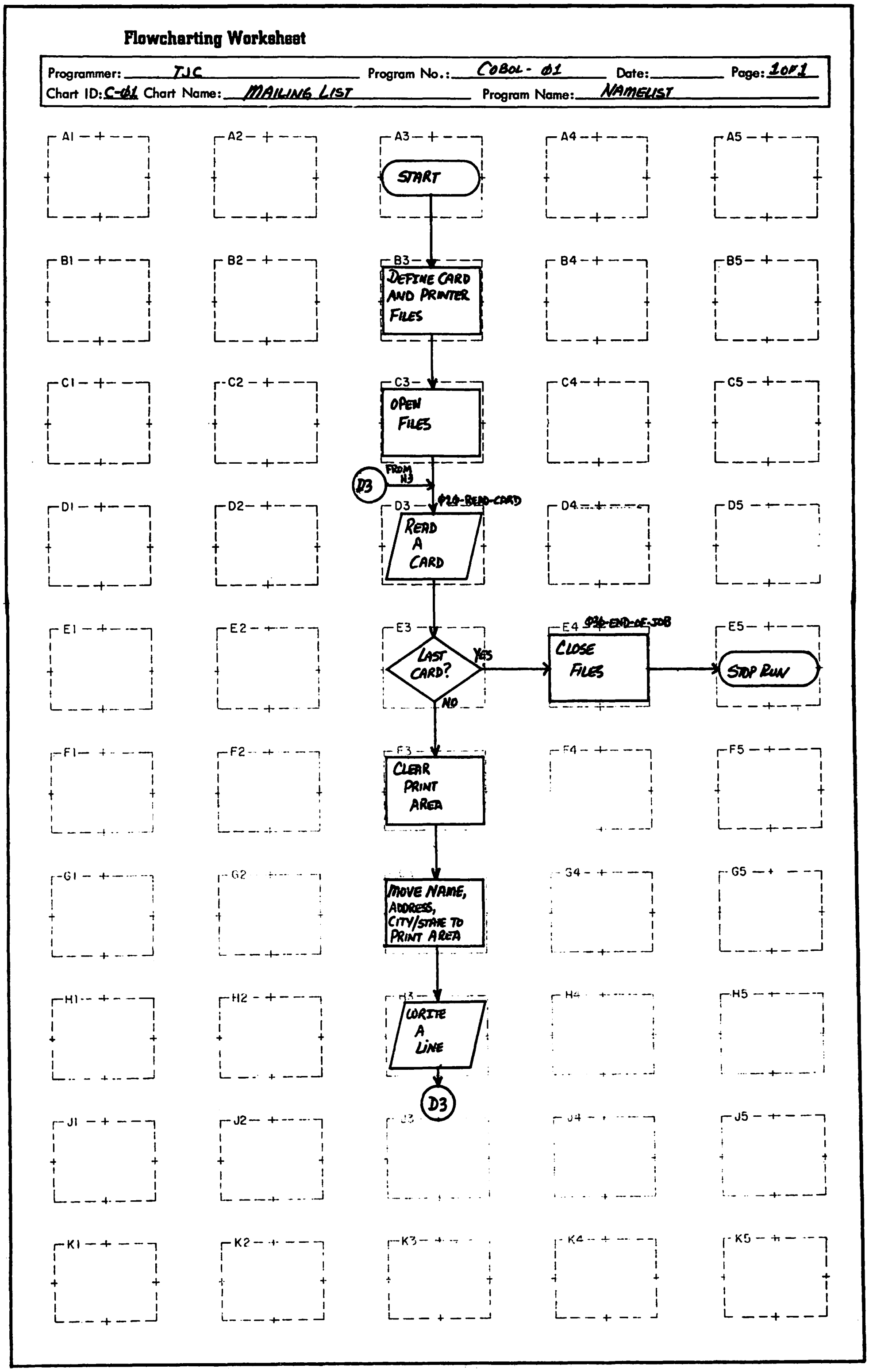
Figure 2-3 Steps in Preparing Report

It can be seen in Figure 2-3 that the data in the card is read into an input area in main storage. The data in the input area is then moved to an output area which is an area reserved for the data which is to be printed on the report. The report is then written.

FLOWCHART

The logic required in the solution of the problem is illustrated in detail in the program flowchart illustrated in Figure 2-4. The basic steps include the following:

1. The format of the Card File and the Printer File are defined in the Data Division of the COBOL program.
2. A COBOL statement is written to OPEN the files and prepare them for processing.
3. A statement is written to Read a Card.
4. Within the structure of the Read Statement, provision is made to check for the last card. This last card may take different forms dependent upon the computer being used. With the System/360, the last card contains the characters /* in card columns 1 and 2. If the last card is not detected, the next instruction in sequence is executed.
5. If it is not the last card, the Print Area (output area) is cleared.
6. The fields to be printed are moved from the Input Area to the Print Area.
7. A statement is written to Write a Line.
8. The computer is then instructed to return to that portion of the program which causes a card to be read. The sequence of operations is then repeated.
9. When the last card is detected, the computer will execute those instructions in the END-OF-JOB routine which includes instructions to Close the Files and Stop the Run.



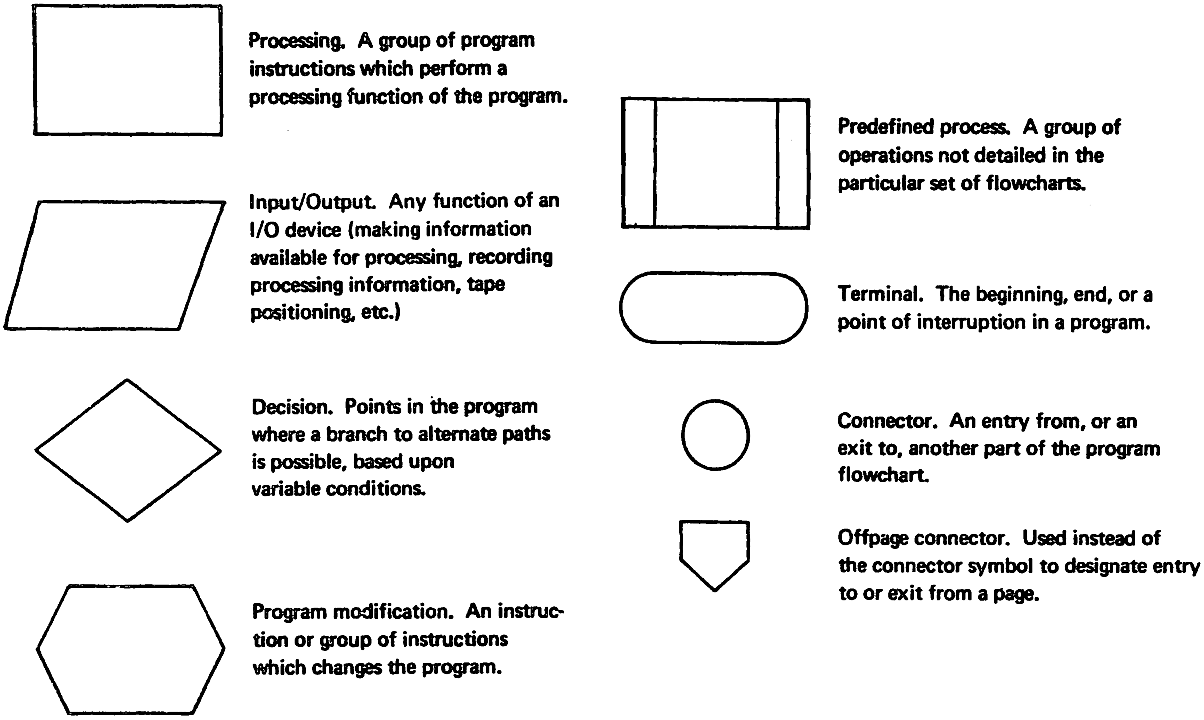
Flowcharting Worksheet

Programmer: TJC Program No.: COBOL-01 Date: Page: 1 OF 1
Chart ID: C-01 Chart Name: MAILING LIST Program Name: NAMELIST

A3
START
B3
DEFINE CARD AND PRINTER FILES
C3
OPEN FILES
D3
FROM H3
020-READ-CARD
D3
READ A CARD
E3
LAST CARD?
YES
NO
E4
030-END-OF-JOB
CLOSE FILES
E5
STOP RUN
F3
CLEAR PRINT AREA
MOVE NAME, ADDRESS, CITY/STATE TO PRINT AREA
H3
WRITE A LINE
D3

Figure 2-4 Program Flowchart

It will be noted in the flowchart in Figure 2-4 that specific symbols are used dependent upon the function to be performed. The following are the commonly used symbols in program flowcharts.



Processing. A group of program instructions which perform a processing function of the program.

Input/Output. Any function of an I/O device (making information available for processing, recording processing information, tape positioning, etc.)

Decision. Points in the program where a branch to alternate paths is possible, based upon variable conditions.

Program modification. An instruction or group of instructions which changes the program.

Predefined process. A group of operations not detailed in the particular set of flowcharts.

Terminal. The beginning, end, or a point of interruption in a program.

Connector. An entry from, or an exit to, another part of the program flowchart.

Offpage connector. Used instead of the connector symbol to designate entry to or exit from a page.

Figure 2-5 Flowchart Symbols

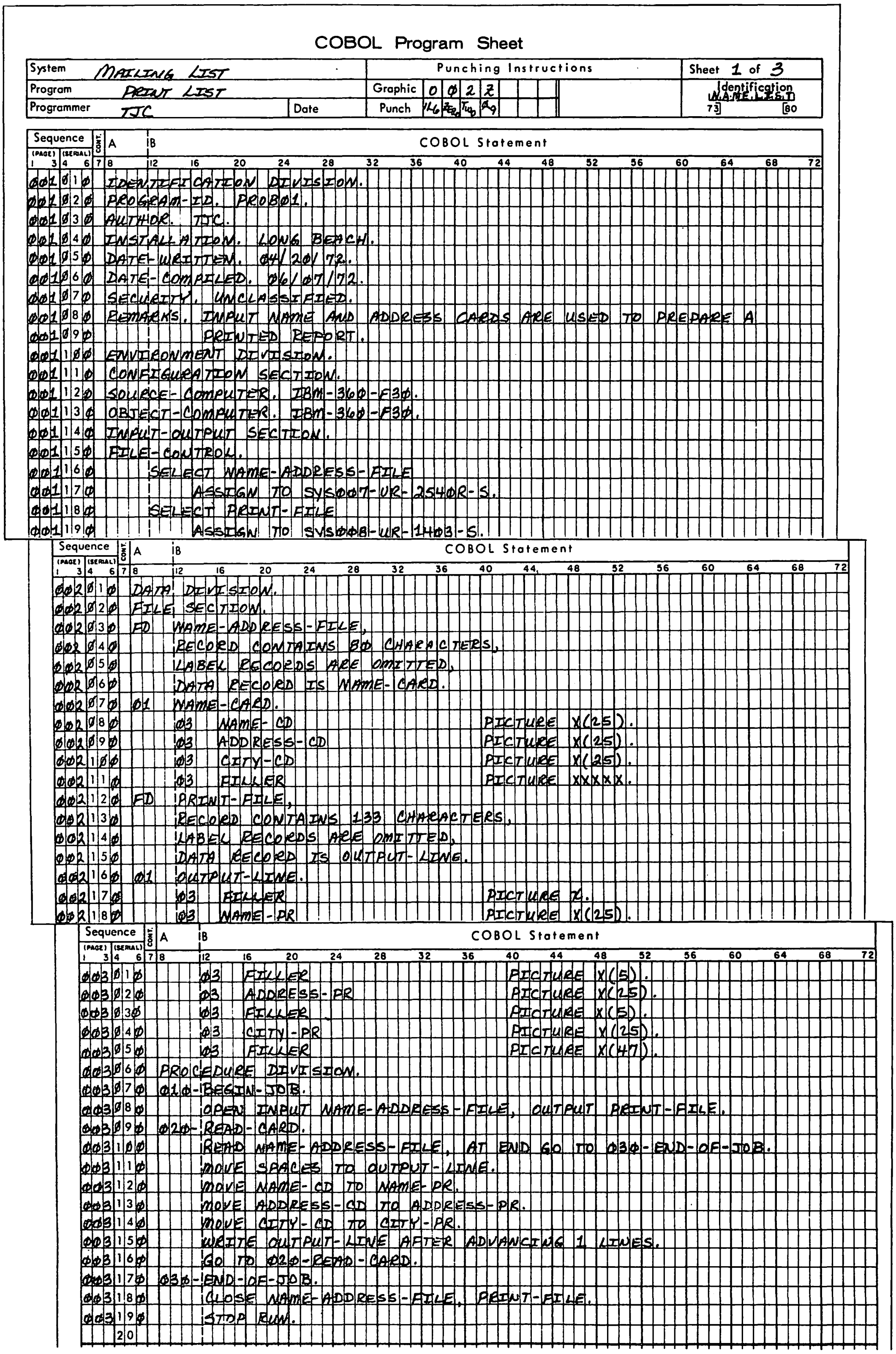
The symbols are drawn using a special "template" which is designed to be used specifically for program flowcharting. This plastic template contains all of the flowcharting symbols which are normally used.

The symbols are usually drawn in one of the numbered blocks on the flowcharting worksheet, as illustrated in Figure 2-4. The blocks are numbered for ease of reference when flowcharts are drawn on separate areas of the flowchart worksheet. As can also be seen from Figure 2-4, each of the symbols are connected by "flowlines" which indicate the sequence of the required operations. Whenever the sequence is to vary from the straightline sequence indicated by the flowlines, "connector" symbols, such as those illustrated in Figure 2-4, are used to indicate the direction of the flow of the logic of the program.

The "normal" direction of flow within a flowchart is always from top–to–bottom and from left–to–right. When the flow within a flowchart follows the "normal" direction of flow, arrowheads such as illustrated in the example in Figure 2–4 may or may not be used, at the discretion of the programmer. The examples in this text will all use arrowheads when the normal direction of flow is used for clarity purposes. Whenever the flow runs from bottom–to–top or from right–to–left, arrowheads must always be used. It should be noted that the flowcharting standards dictate that the arrowheads are to be "open" (↓), not "closed" (↓).

COBOL PROGRAM

The program to read the input data cards and prepare the printed report is illustrated in Figure 2-6. The entries required are explained in detail on the following pages.



COBOL Program Sheet

System MAILING LIST
Program PRINT LIST
Programmer TJC
Date

Punching Instructions

Graphic	O	Ø	2	Z
Punch	[illegible]	Zero	Two	[illegible]

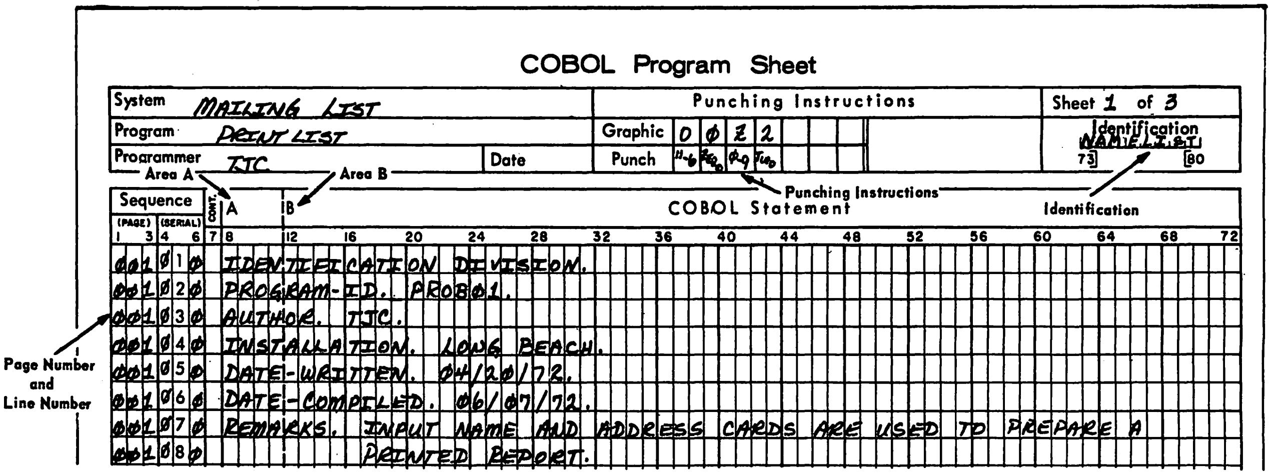
Sheet 1 of 3
Identification NAME LIST 73 80

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. PROB01.
001030 AUTHOR. TJC.
001040 INSTALLATION. LONG BEACH.
001050 DATE-WRITTEN. 04/20/72.
001060 DATE-COMPILED. 06/07/72.
001070 SECURITY. UNCLASSIFIED.
001080 REMARKS. INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A
001090          PRINTED REPORT.
001100 ENVIRONMENT DIVISION.
001110 CONFIGURATION SECTION.
001120 SOURCE-COMPUTER. IBM-360-F30.
001130 OBJECT-COMPUTER. IBM-360-F30.
001140 INPUT-OUTPUT SECTION.
001150 FILE-CONTROL.
001160     SELECT NAME-ADDRESS-FILE
001170         ASSIGN TO SYS007-UR-2540R-S.
001180     SELECT PRINT-FILE
001190         ASSIGN TO SYS008-UR-1403-S.
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  NAME-ADDRESS-FILE,
002040     RECORD CONTAINS 80 CHARACTERS,
002050     LABEL RECORDS ARE OMITTED,
002060     DATA RECORD IS NAME-CARD.
002070 01  NAME-CARD.
002080     03  NAME-CD                 PICTURE  X(25).
002090     03  ADDRESS-CD              PICTURE  X(25).
002100     03  CITY-CD                 PICTURE  X(25).
002110     03  FILLER                  PICTURE  XXXXX.
002120 FD  PRINT-FILE,
002130     RECORD CONTAINS 133 CHARACTERS,
002140     LABEL RECORDS ARE OMITTED,
002150     DATA RECORD IS OUTPUT-LINE.
002160 01  OUTPUT-LINE.
002170     03  FILLER                  PICTURE  X.
002180     03  NAME-PR                 PICTURE  X(25).
003010     03  FILLER                  PICTURE  X(5).
003020     03  ADDRESS-PR              PICTURE  X(25).
003030     03  FILLER                  PICTURE  X(5).
003040     03  CITY-PR                 PICTURE  X(25).
003050     03  FILLER                  PICTURE  X(47).
003060 PROCEDURE DIVISION.
003070 010-BEGIN-JOB.
003080     OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.
003090 020-READ-CARD.
003100     READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.
003110     MOVE SPACES TO OUTPUT-LINE.
003120     MOVE NAME-CD TO NAME-PR.
003130     MOVE ADDRESS-CD TO ADDRESS-PR.
003140     MOVE CITY-CD TO CITY-PR.
003150     WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.
003160     GO TO 020-READ-CARD.
003170 030-END-OF-JOB.
003180     CLOSE NAME-ADDRESS-FILE, PRINT-FILE.
003190     STOP RUN.
   20
```

Figure 2-6 COBOL Program

COBOL Coding Form

The COBOL program is normally written on a special COBOL coding form. A segment of the coding form is illustrated below.



COBOL Program Sheet

System MAILING LIST		Punching Instructions	Sheet 1 of 3
Program PRINT LIST		Graphic O | Ø | Z | 2	Identification NAMELIST
Programmer TJC	Date	Punch 11-6 | Zero | Ø-9 | Two	73 – 80

```
Sequence  A   B
(PAGE)(SERIAL)
ØØ1Ø1Ø IDENTIFICATION DIVISION.
ØØ1Ø2Ø PROGRAM-ID. PROBØ1.
ØØ1Ø3Ø AUTHOR. TJC.
ØØ1Ø4Ø INSTALLATION. LONG BEACH.
ØØ1Ø5Ø DATE-WRITTEN. Ø4/2Ø/72.
ØØ1Ø6Ø DATE-COMPILED. Ø6/Ø7/72.
ØØ1Ø7Ø REMARKS. INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A
ØØ1Ø8Ø                 PRINTED REPORT.
```

Figure 2-7 COBOL Coding Form

At the top of the form space is provided for general information about the program, such as the name of the program, programmer, date, etc. Instructions to the keypunch operator can also be given in the section labeled "Punching Instructions". A character that might create confusion when keypunching can be assigned a special symbol or "graphic" to identify the card code to be punched by the keypunch operator. For example, in the coding sheet illustrated, it can be seen that the graphic "O" is to be punched as an 11-6. Thus "O" is the letter of the alphabet. The graphic "Ø" is to be punched as a zero and is thus a numeric zero when encountered on a coding form. On the right side of the form space is provided to record the Sheet Number and the total number of pages of coding. Below this entry is a section of the form reserved for "Identification". Any numbers, letters of the alphabet, or special characters may be used, including blanks. The "Identification" has no effect on the object program or the compiler even though this information is punched into card columns 73 to 80 of the cards comprising the COBOL program.

Columns 1-6 on the coding form are used to number the lines of coding on the form. Each succeeding line is given a higher number. Columns 1, 2, 3 are used for page number identification, and columns 4, 5, 6 are used to identify each line number (serial number). Columns 4 and 5 are usually prenumbered on the form. Use of card columns 1-6 for sequence numbering program cards is totally optional and is not required by the COBOL compiler. If these columns have been used, the compiler will check the sequence in columns 1-6 of every input card. Every low or equal sequence will produce a warning message on the program listing. However, regardless of the sequence numbers, the compiler will accept program cards in the order in which they actually appear. A sequence error will not disturb a compilation or the execution of a program.

If columns 1-6 are used, PAGE NUMBER (1-3) and SERIAL NUMBER (4-6) must contain only digits. Letters of the alphabet or special characters are not allowed.

Notice that the method of numbering in Figure 2-7 allows additional instructions to be inserted at a later date, if required, without disturbing the ascending sequence of instructions. For example, if an additional line of coding is to be inserted between line 010 and line 020, the line could be identified as line 015 and be inserted into the main program without disturbing the sequence of cards comprising the COBOL program.

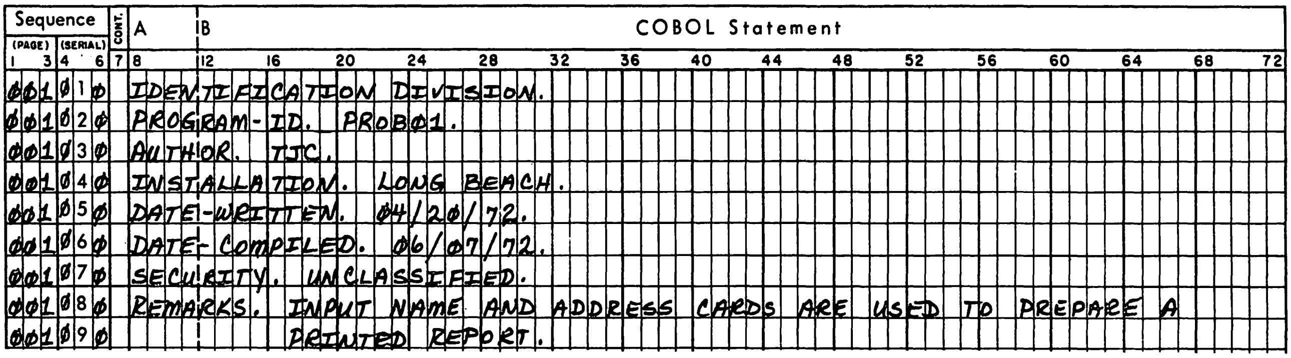
The continuation column (column 7) must be blank on all cards except when a special type of COBOL word called a non-numeric literal must be continued on a second line. This use of the continuation column is explained in detail later in the text.

Columns 8-72 are used for program entries. The columns are grouped into two areas, AREA A (column 8-11) and AREA B (column 12-72). Certain portions of the COBOL program must begin in Area A and certain portions must begin in Area B. The specific rules for recording the various entries are discussed in detail throughout this chapter.

IDENTIFICATION DIVISION

The Identification Division of a COBOL program is normally written first. This Division specifies information which serves to identify and document the program. The following is an illustration of the Identification Division of the COBOL program used to prepare the Name and Address Listing.

EXAMPLE



Sequence
(PAGE)
(SERIAL)
CONT.
A
B
COBOL Statement
1 3 4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID.  PROB01.
001030 AUTHOR.  TJC.
001040 INSTALLATION.  LONG BEACH.
001050 DATE-WRITTEN.  04/20/72.
001060 DATE-COMPILED.  06/07/72.
001070 SECURITY.  UNCLASSIFIED.
001080 REMARKS.  INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A
001090          PRINTED REPORT.
```

Figure 2-8 Identification Division

When writing a COBOL program, the method of recording the Division Names and related entries, the spacing, and the presence or absence of the period is extremely important. The following statements summarize the rules for recording the entries in the Identification Division.

1. The beginning of the Identification Division is marked by the "Division Header Entry" which consists of the name IDENTIFICATION DIVISION. A division header entry always begins on a line by itself, and must begin in Area A, and be followed by a period.

2. The Identification Division is made up of "Paragraph Headers". These Paragraph Headers consist of a series of fixed names (PROGRAM-ID, AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY, and REMARKS). These paragraph names must begin in Area A and be followed by a period and a space.

3. The words or sentences following the Paragraph Headers must be contained within Area B.

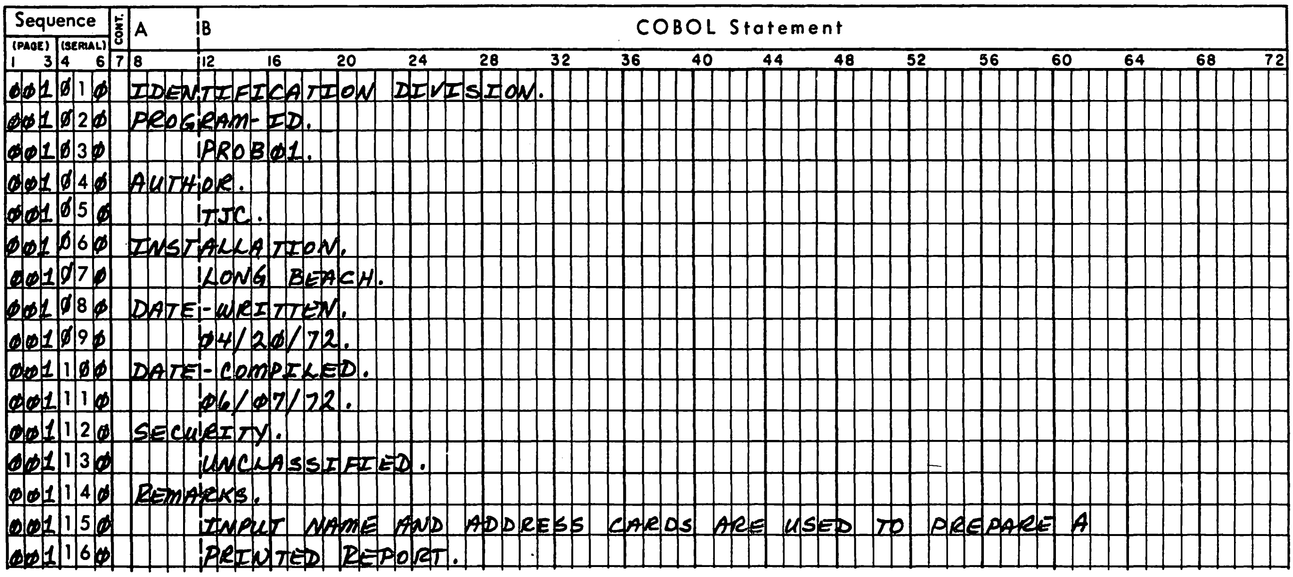
4. The Program Name following PROGRAM-ID must consist of alphabetic and numeric characters, the first of which should be alphabetic. The first eight characters of the Program Name identify the object program.

5. The words following the paragraph header entries AUTHOR, INSTALLATION. DATE-WRITTEN, DATE-COMPILED, SECURITY, and REMARKS, may be originated at the discretion of the programmer and may contain any characters in the EBCDIC set. If sentences are written, they must be contained within Area B. Coding may not be extended into columns 73-80. The programmer may break off the coding anywhere before column 73 and record the following entries on the next line if they are contained within Area B.

6. Each of the entries in the Identification Division must be terminated by a period.

The following example illustrates an alternative method of recording the entries for the Identification Division on the COBOL coding form. Note that the Program Name and the words and sentences following the paragraph headers are recorded on individual lines.

EXAMPLE


Sequence
(PAGE)
(SERIAL)
CONT.
A
B
COBOL Statement
1 3 4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID.
001030     PROB01.
001040 AUTHOR.
001050     TJC.
001060 INSTALLATION.
001070     LONG BEACH.
001080 DATE-WRITTEN.
001090     04/20/72.
001100 DATE-COMPILED.
001110     06/07/72.
001120 SECURITY.
001130     UNCLASSIFIED.
001140 REMARKS.
001150     INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A
001160     PRINTED REPORT.
```

Figure 2-9 Identification Division with Entries on Separate Lines

FORMAT NOTATION

Technical reference manuals developed for the COBOL programming language provide a standard format notation for the various elements of COBOL. These generalized descriptions are intended to guide the programmer in writing COBOL programs. The format notation for the Identification Division is illustrated below.

```
IDENTIFICATION DIVISION.

PROGRAM-ID.  program-name.

[AUTHOR.  [comment-entry]...]

[INSTALLATION.  [comment-entry]...]

[DATE-WRITTEN.  [comment-entry]...]

[DATE-COMPILED.  [comment-entry]...]

[SECURITY.  [comment-entry]...]

[REMARKS.  [comment-entry]...]
```

Figure 2-10 Format Notation for Identification Division

It is imperative that the COBOL programmer thoroughly understand this system of notation. The basic rules as related to the Identification Division are summarized below.

1. All words printed entirely in capital letters are RESERVED words. These words have preassigned meanings in the COBOL language and are not to be used for any other purpose. In all formats, words written in capital letters selected for use must be duplicated exactly.

2. All underlined reserved words are required unless the portion of the format containing them is itself optional. These are key words. If any such word is missing or is incorrectly spelled, it is considered an error in the program.

3. Lower case words in formats represent information that must be supplied by the programmer.

4. Punctuation is essential where it is shown.

5. Square brackets ([ ]) are used to indicate that the enclosed item may be used or omitted, depending on the requirements of the particular program.

By applying the above rules of format notation to the Identification Division, it can be seen that AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY, and REMARKS are enclosed in brackets and are optional, that is, these paragraphs are not a required portion of the Identification Division. Note, therefore, that the only required entry in the Identification Division is the division header, IDENTIFICATION DIVISION, and the PROGRAM-ID.

SUMMARY - IDENTIFICATION DIVISION

It should be again noted that there are two areas on the COBOL coding sheet, Area A (columns 8-11) and Area B (columns 12-72).

The division-name, IDENTIFICATION DIVISION, must begin in Area A, and be followed by a period. This entry must appear on a line by itself.

The paragraph-names PROGRAM-ID, AUTHOR, DATE-WRITTEN, DATE-COMPILED, SECURITY, and REMARKS must also begin in Area A and must be followed immediately by a period and then a space. Sentences within a paragraph may start on the same line as the paragraph-name or on a separate line. Succeeding lines of the paragraph must begin within Area B.

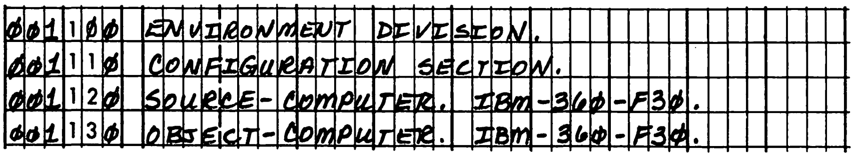
ENVIRONMENT DIVISION

The Environment Division contains two "Sections", the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION. The Configuration Section identifies the computer on which the program is to be compiled and executed, and the Input-Output Section assigns files to input-output devices and may specify special input/output techniques.

Configuration Section

The following segment of the COBOL coding form illustrates the suggested entries for the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

EXAMPLE



```
001100 ENVIRONMENT DIVISION.
001110 CONFIGURATION SECTION.
001120 SOURCE-COMPUTER. IBM-360-F30.
001130 OBJECT-COMPUTER. IBM-360-F30.
```

Figure 2-11 Environment Division – Configuration Section

The Division Header, ENVIRONMENT DIVISION, the Section Header, CONFIGURATION SECTION, and the Paragraph Headers, SOURCE-COMPUTER, OBJECT-COMPUTER, must begin in Area A and be followed by a period. There must always be at least one blank space following a period. The entries following Source-Computer and Object-Computer must be contained within Area B.

The SOURCE-COMPUTER paragraph names the computer on which the COBOL compiler is to be run to prepare the Object Program. The OBJECT-COMPUTER paragraph identifies the computer on which the Object Program is to be run to process the data. In the sample program both the Source-Computer and Object-Computer are specified as IBM-360-F30. The F30 specifies the model number (Model 30) of the System/360. The letter of the alphabet preceding the model number specifies the storage capacity of the system and is based upon the following chart.

C – 8,192 bytes
D – 16,384 bytes
E – 32,768 bytes
F – 65,536 bytes
G – 131,072 bytes
H – 262,144 bytes
I – 524,288 bytes

From the coding it can be seen that an IBM System/360, Model 30 with 65,536 bytes of storage is to be used to compile and execute the program. It should be noted that the information contained for the Source-Computer and the Object-Computer are merely for documentation purposes and any values desired may be specified to identify the computer.

The COBOL format notation for the Configuration Section is illustrated below.

```
ENVIRONMENT DIVISION.

CCNFIGURATICN SECTICN.
SOURCE-CCMPUTER. scurce-computer-entry
OBJECT-COMPUTER.  object-computer-entry
```

Figure 2-12 Configuration Section – Format Notation

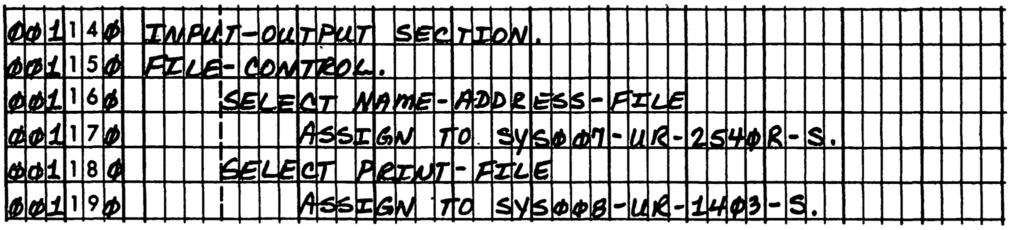
In the format notation illustrated above, it can be seen that the terms CONFIGURATION SECTION, SOURCE-COMPUTER, and OBJECT-COMPUTER must be included as shown followed by the entries which describe the source and object computers.

INPUT-OUTPUT SECTION

The INPUT-OUTPUT SECTION of the Environment Division is divided into two paragraphs: the FILE-CONTROL paragraph and the I-O-CONTROL paragraph. The File-Control paragraph names and associates the files to be processed with the input/output devices being used. In the sample problem, the Name and Address Card File is to be read using the IBM 2540 Card Reader. The printed report to be prepared (output file) is to be produced using the IBM 1403 printer.

The I-O-Control paragraph defines special input/output techniques. The following example illustrates the suggested entries for the INPUT-OUTPUT SECTION of the Environment Division.

EXAMPLE



```
001140 INPUT-OUTPUT SECTION.
001150 FILE-CONTROL.
001160     SELECT NAME-ADDRESS-FILE
001170         ASSIGN TO SYS007-UR-2540R-S.
001180     SELECT PRINT-FILE
001190         ASSIGN TO SYS008-UR-1403-S.
```

Figure 2-13 Input-Output Section

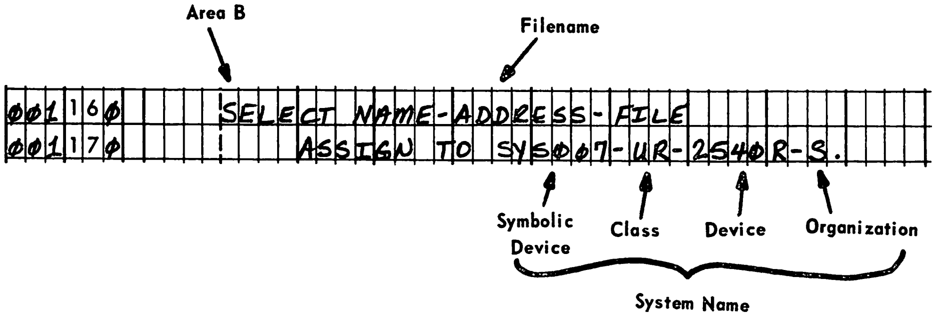
The general format of the FILE-CONTROL paragraph within the Input-Output Section is illustrated in Figure 2-14.

```
FILE-CONTROL.

    {SELECT Clause
    ASSIGN Clause
    [RESERVE Clause]
    [FILE-LIMIT Clause]
    [ACCESS MODE Clause]
    [PROCESSING MODE Clause]
    [ACTUAL KEY Clause]
```

Figure 2-14 File-Control Paragraph – Format Notation

The only entries required in the sample program for the INPUT-OUTPUT SECTION are the FILE-CONTROL paragraph, consisting of the SELECT clause and the ASSIGN clause. The SELECT clause is used to name each file in a program and the ASSIGN clause is used to assign a file on an external device on the computer system. The SELECT and ASSIGN clauses used for the card input file in the sample program are illustrated below.

EXAMPLE



```
001160    SELECT NAME-ADDRESS-FILE
001170        ASSIGN TO SYS007-UR-2540R-S.
```

Figure 2-15 Select Clause and Assign Clause

The SELECT clause must begin with the words SELECT and be followed by a "file-name". A file-name should be assigned that is descriptive of the file being processed. The "file-name" is a name originated by the programmer and may be composed of a combination of not more than 30 characters chosen from the following set of 37 characters:

0 through 9 (numeric digits)
A through Z (alphabetic letters)
- (hyphen)

File-names must not begin or end with a hyphen but a hyphen may be contained within the name. The end of a File Name is detected by the COBOL compiler by the space at the end of the name. Therefore, no blanks are permitted within the name. The name of each file must be unique within the program.

Note in Figure 2-15 that the name assigned to the Name and Address Card file is "NAME-ADDRESS-FILE". It should be noted that this name was chosen because it was descriptive of the file to be processed.

The ASSIGN clause within the SELECT clause is used to relate a file to a particular Input/Output device. In the example in Figure 2-15, it can be seen that the words ASSIGN TO begin the Assign clause. The word ASSIGN is required in the clause and the word TO may be used at the option of the programmer. It is not required.

The SYSTEM NAME is a grouping of information which specifies the type of device which is to be used for the file. It consists of the Symbolic Device entry, the Class entry, the Device entry, and the Organization entry.

The External Name is a name which must begin with the letters of the alphabet SYS. It is used by the Operating System to associate a particular device with the file which is defined within the program. It should be noted that this entry is required when ANSI COBOL is to be used with the IBM Disk Operating System. When, however, the program is to be compiled on other machines or under other operating systems, this entry may vary. In the example in Figure 2-15 it can be seen that the external name SYS007 is used because this is the external name which is associated with the proper card reader by the operating system.

The class entry is used to specify the device class. The entry "UR" is used for a card reader and stands for Unit-Record. Card readers and printers are always defined as Unit-Record devices and must have the class entry UR.

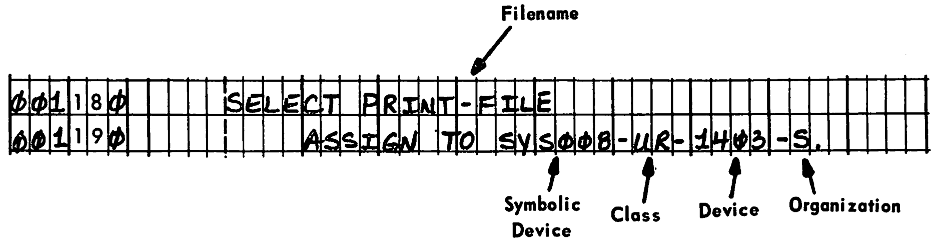
The Device Entry specifies the type of device which is to be used for the file. The allowable device-numbers for unit record devices are: 1442R, 1442P, 1403, 1404 (continuous forms only), 1443, 1445, 2501, 2520P, 2540R, and 2540P. Note in the example in Figure 2-15 that the entry 2540R is specified because the card file is to be read from a 2540 card reader. The "R" in the device entry specifies that a card reader is to be used. The "P" specifies the card punch for the same device. Again, these are IBM devices and may vary dependent upon the computer being used.

The Organization entry is used to indicate the file organization. In the example, it can be seen the entry is the letter of the alphabet "S", which indicates a sequential organization. All files which utilize unit-record devices must be organized in a sequential manner.

It should also be noted in Figure 2-15 that the Select Clause (line 160) for the Name-Address-File is continued on line 170. This line must begin within Area B. In the example, the "ASSIGN TO . . ." line was indented to column 16 for readability, but it could have begun in column 12 if desired.

The SELECT and ASSIGN clauses for the PRINT-FILE are illustrated in Figure 2-16.

EXAMPLE



```
001180     SELECT PRINT-FILE
001190         ASSIGN TO SYS008-UR-1403-S.
```

Figure 2-16 Select Clause and Assign Clause

As can be seen from the example in Figure 2-16, the same format is used for the printer output file as is used for the card input file. The sentence must always start with the word SELECT. The File-name used for the printer output file is PRINT-FILE which, again, is a name which denotes the file which it names.

The Symbolic Device entry specifies that the device to be used for the file will be assigned to the symbolic name SYS008 by the Disk Operating System. As noted previously, a printer is always a unit-record class of device and, therefore, the UR entry is specified for the class entry.

The type of printer to be used for the report is an IBM 1403 so the value 1403 is entered for the Device entry to indicate to the COBOL compiler that the file is to be designed for use with a 1403 printer. The organization on a printer is always sequential (S).

The SELECT and ASSIGN clauses are used to give a name to a file and to indicate the type of hardware which is to be used for the file. It remains, however, to indicate further attributes of the file such as the length of the records in the file and the formats of the records which are utilized in the file processing. These definitions are made in the DATA DIVISION of the COBOL program.

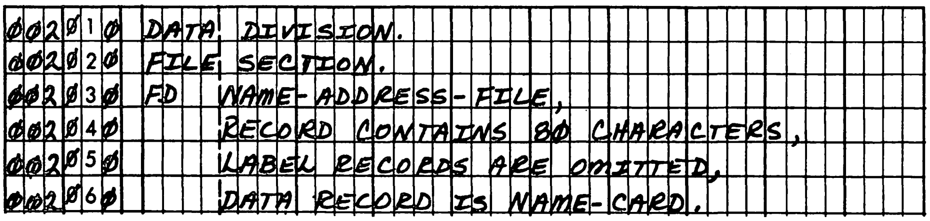
DATA DIVISION

The DATA DIVISION of a COBOL source program describes the files to be processed by the object program. The Data Division is subdivided into sections. In the sample problem introduced in this chapter, only the FILE SECTION of the DATA DIVISION is used.

The FILE SECTION describes the content and organization of the files to be processed. In addition to describing the files to be processed, RECORD DESCRIPTION entries are required which describe the individual fields contained in a data record of a file.

The first segment of the File Section of the Data Division used in the sample program consists of the entries illustrated below.

EXAMPLE



```
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  NAME-ADDRESS-FILE,
002040     RECORD CONTAINS 80 CHARACTERS,
002050     LABEL RECORDS ARE OMITTED,
002060     DATA RECORD IS NAME-CARD.
```

Figure 2-17 Data Division – File-Section Entries

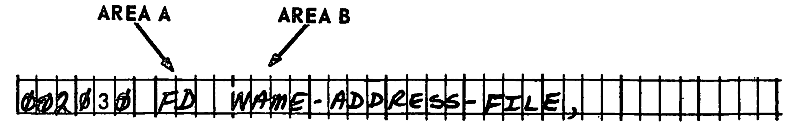
The entries illustrated in Figure 2-17 are used to provide a general description of the contents and organization of the records contained in the file. The format of the entries used in the sample problem in the first section of the Data Division are illustrated below.

```
FD file-name
   [RECORD CONTAINS Clause]
   LABEL RECCRDS Clause
   [DATA RECORDS Clause]
```

Figure 2-18 File-Section – Format Notation

The FD entry (File Description) describes each data file (for example, the card file) to be processed by the object program. The name following the FD must be the file-name used in the Select Sentence of the File Control Paragraph (see Figure 2-15). The FD entry must begin in Area A and the file-name must begin in Area B.

EXAMPLE



```
002030 FD  NAME-ADDRESS-FILE,
```

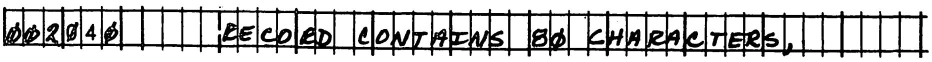
Figure 2-19 FD Entry

All subsequent entries following the FD entry must begin in Area B.

RECORD CONTAINS Clause

The RECORD CONTAINS clause is used to specify the maximum size of logical records. An example is illustrated below.

EXAMPLE



```
002040     RECORD CONTAINS 80 CHARACTERS.
```

```
RECORD CONTAINS [integer-1 TC] integer-2 CHARACTERS
```

Figure 2-20 RECORD CONTAINS Clause

Integer-1 and Integer-2 are used to specify minimum and maximum record sizes respectively. In the sample program the entry RECORD CONTAINS 80 CHARACTERS is specified.

The RECORD CONTAINS clause is not necessary for a file having equal length records, but the entry is suggested for documentation purposes.

Record lengths are determined by the compiler regardless of whether or not the clause is specified.

LABEL RECORDS Clause

This clause specifies the presence of standard or non-standard labels on a file, or the absence of labels. Labels are used when processing data stored on magnetic tape and magnetic disks. An example of the entry used in the sample program is illustrated below.

EXAMPLE

```
002050     LABEL RECORDS ARE OMITTED.
```

```
       ( RECORD IS   )   ( OMITTED                          )
LABEL  {             }   { STANDARD                         }
       ( RECORDS ARE )   ( data-name-1 [data-name-2] ...    )
```

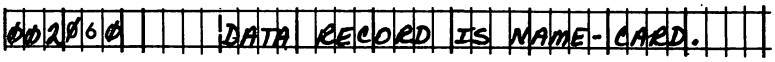
Figure 2-21 LABEL RECORDS Clause

The OMITTED option MUST be specified for a file assigned to a unit record device. Thus, the entry in the sample program LABEL RECORDS ARE OMITTED is required.

DATA RECORD Clause

This clause specifies the name of the logical records within a file.

EXAMPLE



```
002060     DATA RECORD IS NAME-CARD.
```

$$\underline{\text{DATA}} \begin{Bmatrix} \underline{\text{RECORD}} \text{ IS} \\ \underline{\text{RECORDS}} \text{ ARE} \end{Bmatrix} \text{ data-name-1 [data-name-2] ...}$$

Figure 2-22 Example of DATA RECORDS Clause

The entry for "data-name-1" . . . in the DATA RECORDS Clause is for the purpose of assigning a symbolic name to the individual records (cards) of a file. In the sample problem, the entry DATA RECORD IS NAME-CARD is specified. Thus, NAME-CARD may be used in subsequent sections of the program to reference the individual cards to be processed. It should be noted that more than one type of record may be contained within a single file and the "data-name-2", etc. entries are used to give names to all of the different types of records which may be found in a file.

The formation of a "data-name" follows the rules of data-name formation, that is, the name must contain at least one alphabetic character and be composed of not more than 30 characters. The characters may be numbers, letters of the alphabet, or the hyphen. The hyphen must not be the first or last character of the name.

At this point, in the COBOL program, the programmer has assigned a symbolic name for the entire File, and a symbolic name for each record within the File.

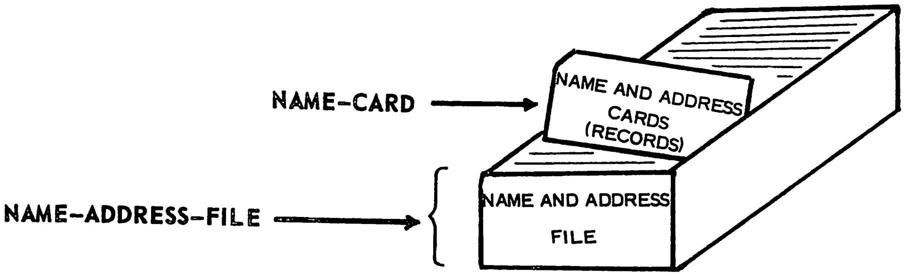


Figure 2-23 Illustration of File and Records within a File

RECORD DESCRIPTION

After the file has been described by the RECORD CONTAINS, LABEL RECORDS ARE, and DATA RECORD IS Clauses, the individual records within the file are described. The following segment of the COBOL program developed in this chapter illustrates the entries to describe the individual records (cards) to be processed.

EXAMPLE

```
Sequence      A   B                              COBOL Statement
(PAGE)(SERIAL)
1  3 4  6 7 8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  NAME-ADDRESS-FILE,
002040     RECORD CONTAINS 80 CHARACTERS,
002050     LABEL RECORDS ARE OMITTED,
002060     DATA RECORD IS NAME-CARD.
002070 01  NAME-CARD.
002080     03  NAME-CD                 PICTURE X(25).
002090     03  ADDRESS-CD              PICTURE X(25).
002100     03  CITY-CD                 PICTURE X(25).
002110     03  FILLER                  PICTURE XXXXX.
```

Figure 2-24 Record Description Entries

A Record Description entry specifies the characteristics of each item in a data record. Every item must be described in a separate entry in the same order in which the item appears in the record. Each Record Description entry consists of a level-number, a data-name, and a series of independent clauses followed by a period.

The general format of a Record Description entry is:

```
                   ( data-name )
level number      {             }
                   ( FILLER    )

    [REDEFINES Clause]
    [BLANK WHEN ZERO Clause]
    [JUSTIFIED Clause]
    [OCCURS Clause]
    [PICTURE Clause]
    [SYNCHRONIZED Clause]
    [USAGE Clause]
    [VALUE Clause].
```

Figure 2-25 Record Description Entries – Format Notation

LEVEL INDICATORS

"Level Indicators" are used to show how data items are related to each other. The most inclusive grouping of data is the file. The level indicator for the file is "FD", which has been previously described. Level number 01 is used to specify a logical record. Thus, the individual card (logical record) is identified by a 01 level indicator.

In the sample program, the individual logical record of the file is identified by the entry "01 NAME-CARD".

EXAMPLE

```
002060     DATA RECORD IS NAME-CARD.
002070 01  NAME-CARD.
```

```
level number { data-name }
             { FILLER    }
```

Figure 2-26 Record Description Entry

Note in the example in Figure 2-26 that the level-number 01 is specified and the data-name NAME-CARD is specified in Area B. All 01 level indicators must be specified in Area A.

Subordinate data items (fields) that constitute a logical record are grouped in a hierarchy, and identified with level numbers 02 to 49. Thus, fields within the record are identified by additional level numbers. Note that in the sample program level number 03 is used to identify the fields within a record. Any number higher than 01 could have been selected. Identifying the fields within a record with a level number of 03 allows further subdivision of the fields if required at some time in the future.

The following example illustrates the entries to describe the fields within "NAME-CARD".

EXAMPLE

```
002060     DATA RECORD IS NAME-CARD.
002070 01  NAME-CARD.
002080     03  NAME-CD                 PICTURE X(25).
002090     03  ADDRESS-CD              PICTURE X(25).
002100     03  CITY-CD                 PICTURE X(25).
002110     03  FILLER                  PICTURE XXXXX.
```

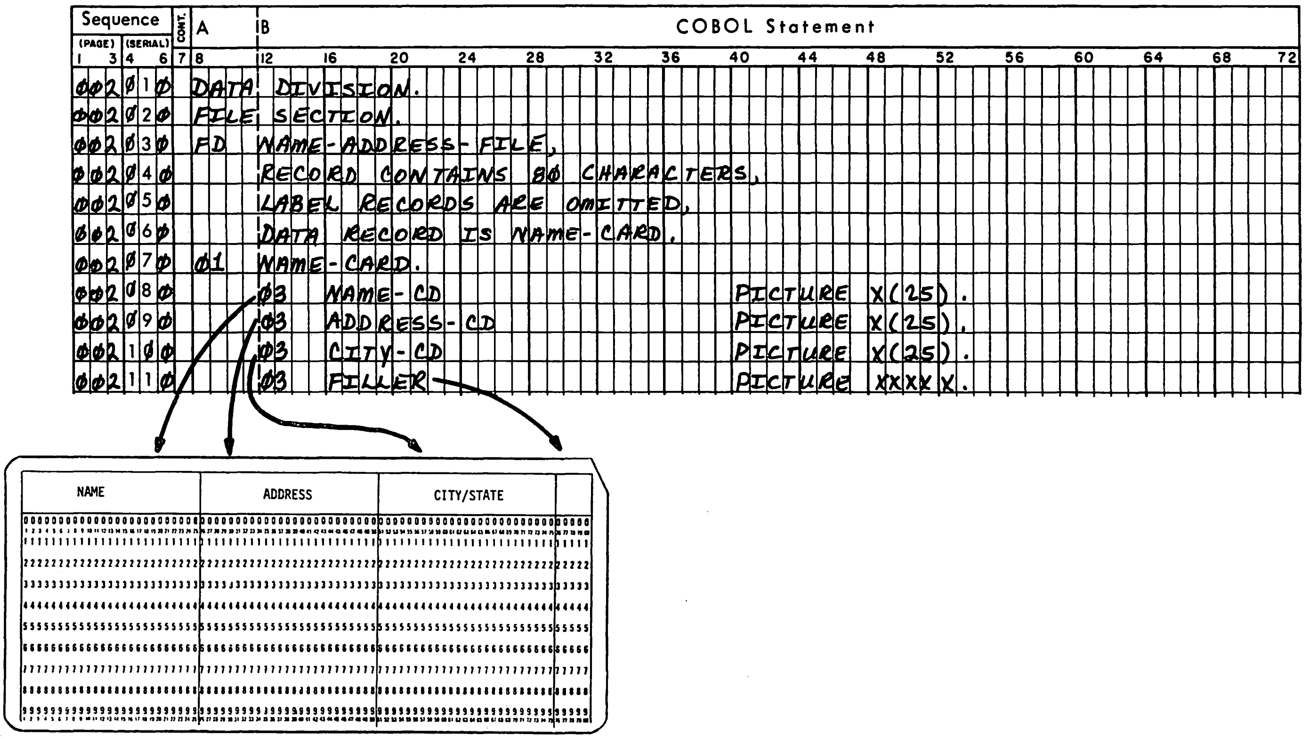
Figure 2-27 Entries to Describe Fields Within a Record

The entries on lines 080, 090, 100, and 110 in Figure 2-27 are used to describe the fields within the card referenced by the symbolic name "NAME-CARD".

When describing the fields within a record, a "data-name" or the COBOL word FILLER must be specified following the level number. The Data-Name selected may contain up to 30 characters. The construction of the name follows the same rules as those for the formation of file-names. Note in the sample program that the Data-Names NAME-CD, ADDRESS-CD, and CITY-CD have been selected to reference the corresponding fields on the card to be processed. These names have been chosen because they represent the content of each field within the card. The suffix "CD" is used to indicate that the names refer to fields within the CarD record. It is normally a good practice to choose a data-name which indicates the contents of the field and to also use a prefix or a suffix, such as in the example, to indicate the source of the field.

If some of the characters in a record are not used in the processing steps of a program, then the data description of these characters need not include a data-name. In this case, FILLER is written in lieu of a data-name after the level number. Note the use of the FILLER in the illustration below.

EXAMPLE



```
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  NAME-ADDRESS-FILE,
002040     RECORD CONTAINS 80 CHARACTERS,
002050     LABEL RECORDS ARE OMITTED,
002060     DATA RECORD IS NAME-CARD.
002070 01  NAME-CARD.
002080     03  NAME-CD                 PICTURE X(25).
002090     03  ADDRESS-CD              PICTURE X(25).
002100     03  CITY-CD                 PICTURE X(25).
002110     03  FILLER                  PICTURE XXXXX.
```

Figure 2-28 Illustration of Card Format and Record Description Entries

It should be noted that subdivisions of a record that are not themselves subdivided are called ELEMENTARY ITEMS. Thus, from the example above, the fields with the data names NAME-CD, ADDRESS-CD, and CITY-CD are called Elementary Items.

PICTURE Clause

Associated with each Elementary Item (entry following level 03 in the sample problem) there must also be an associated PICTURE Clause.

The PICTURE Clause specifies a detailed description of an elementary level data item. This description indicates whether a field contains letters of the alphabet only, letters of the alphabet, numbers, and special symbols, or numbers only.

EXAMPLE

PICTURE CLAUSES

```
002060     DATA RECORD IS NAME-CARD.
002070  01 NAME-CARD.
002080     03  NAME-CD                PICTURE X(25).
002090     03  ADDRESS-CD             PICTURE X(25).
002100     03  CITY-CD                PICTURE X(25).
002110     03  FILLER                 PICTURE XXXXX.
```

Figure 2-29 Record Description Entries with PICTURE Clauses

The general format of the PICTURE Clause is:

```
{ PICTURE }
{         }  IS character string
{ PIC     }
```

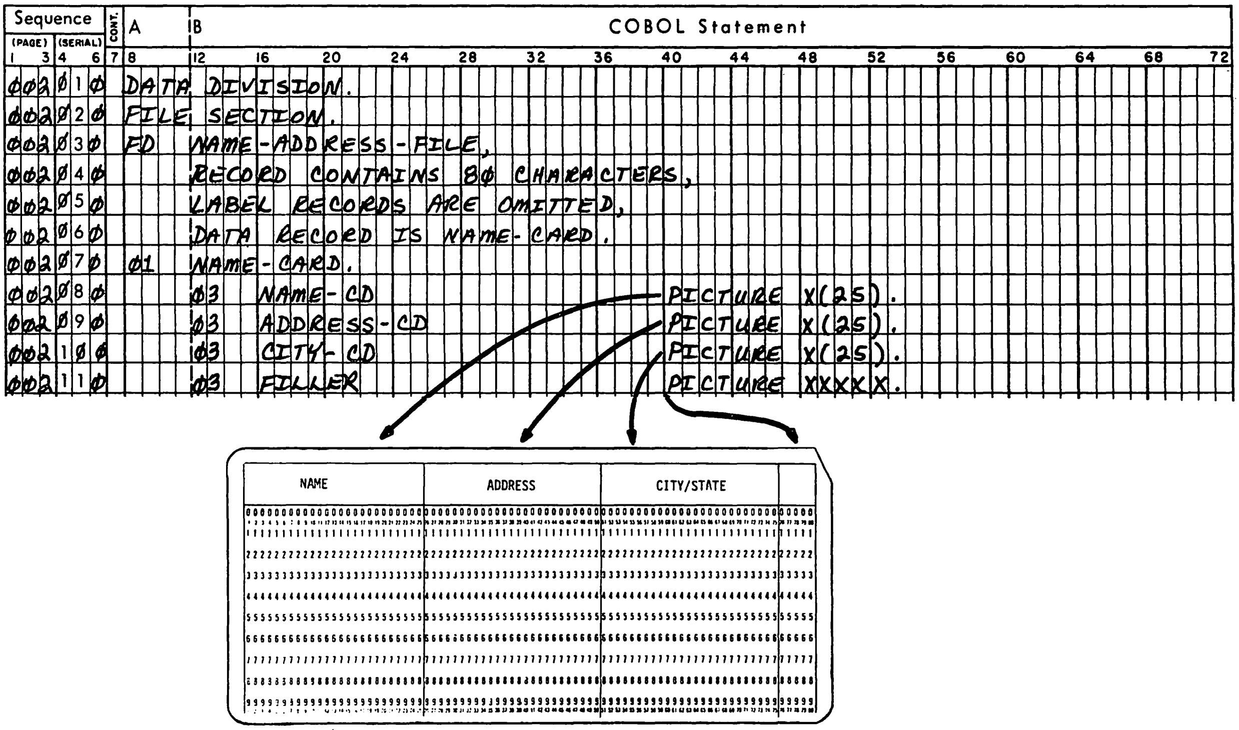
Figure 2-30 PICTURE Clause – Format Notation

The character string referenced in the Format Notation consists of certain allowable combinations of characters in the COBOL character set. These allowable combinations determine the category of the elementary item. There are five catagories of data that can be described with a PICTURE Clause. In the sample program, the data is Alphanumeric. The Alphanumeric category uses a notation of "X" following the word PICTURE. A Picture with a notation of "X" indicates that the field being described may contain any of the characters which may be available on the computer.

Describing the Input Card

The following diagram illustrates an input data card and the segment of the entries in the Data Division describing the card. Note the relationship of the card and the Picture Clauses.

EXAMPLE



```
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  NAME-ADDRESS-FILE,
002040         RECORD CONTAINS 80 CHARACTERS,
002050         LABEL RECORDS ARE OMITTED,
002060         DATA RECORD IS NAME-CARD.
002070 01  NAME-CARD.
002080     03  NAME-CD                 PICTURE X(25).
002090     03  ADDRESS-CD              PICTURE X(25).
002100     03  CITY-CD                 PICTURE X(25).
002110     03  FILLER                  PICTURE XXXXX.
```

Figure 2-31 Picture Clause with X Notation

Note, in the example above, that the field referenced by "NAME-CD" is specified as PICTURE X(25). This entry signifies that the NAME-CD field will occupy the first 25 card columns of the card as signified by the X(25). As the entry following the word PICTURE consists of X's, any allowable characters on the computer could be present in this field. This entry could have been specified as PICTURE followed by 25 individual X's if desired. The use of the 25 in parentheses provides a convenient method for representing 25 X's and greatly reduces the amount of writing required. The word PICTURE begins in column 40 in the sample program. It is recorded in this position for clarity of reading and could have begun one blank space following the related data-name if desired.

The entries to describe the ADDRESS-CD field and the CITY-CD field are identified with a PICTURE X(25). This notation indicates that both of these fields are 25 card columns in length and may consist of any of the characters which can be stored in the computer.

The final entry in this segment of the program consists of a FILLER followed by a picture "PICTURE XXXXX". This entry specifies that card columns 76 through 80 of the card are not to be used or referenced in this program.

It should be noted also from the format notation illustrated in Figure 2-30 that the word PICTURE need not be spelled out completely. It is perfectly valid to use the abbreviation PIC, instead of the entire word. If this were done, the record description for the card input data would appear as illustrated below.

EXAMPLE

```
002070 01  NAME-CARD.
002080         03  NAME-CD                     PIC X(25).
002090         03  ADDRESS-CD                  PIC X(25).
002100         03  CITY-CD                     PIC X(25).
002110         03  FILLER                      PIC XXXXX.
```

Figure 2-32 Example of PIC Entry

Note from the above example that the only change from the example in Figure 2-31 is that the value "PIC" is used instead of the entire word "PICTURE". The use of this abbreviation is intended to save coding and writing time for the programmer.

Describing the Printed Report

The next entry in the sample program is used to define the printed report that is to be prepared (output file).

EXAMPLE

```
002120 FD  PRINT-FILE,
002130     RECORD CONTAINS 133 CHARACTERS,
002140     LABEL RECORDS ARE OMITTED,
002150     DATA RECORD IS OUTPUT-LINE.
002160 01  OUTPUT-LINE.
002170     03  FILLER                  PICTURE X.
002180     03  NAME-PR                 PICTURE X(25).
003010     03  FILLER                  PICTURE X(5).
003020     03  ADDRESS-PR              PICTURE X(25).
003030     03  FILLER                  PICTURE X(5).
003040     03  CITY-PR                 PICTURE X(25).
003050     03  FILLER                  PICTURE X(47).
```

Figure 2-33 FD Entry for "Print-File"

The name following the FD entry must be the same as the name specified in the Select clause of the File Control paragraph. This is the name that references the entire printed output file.

Note the similarity of the entries following FD PRINT-FILE to the entries previously explained and related to the input card file.

The RECORDS CONTAINS clause is specified as 133. The maximum number of print positions available with the IBM 1403 Printer used in the example is 132. One additional position is specified to allow for a special carriage control character which is placed at the beginning of each print line and used by the COBOL compiler to control single, double spacing, etc. Thus, 132 print positions plus 1 additional position required for carriage control functions equals the "133" specified in the RECORD CONTAINS clause.

It is suggested that when programming using COBOL as many standarized techniques be utilized as possible. Throughout this text all print lines will be specified as containing 133 positions.

The symbolic name of the individual line to be printed is identified by a 01 Level Indicator. The programmer assigned name to the print line in the sample program is OUTPUT-LINE.

The 03 Level Indicators below the entry for OUTPUT-LINE are used to describe the format of the output report. The first 03 Level entry consists of a one position FILLER and is necessary when using output in the form of a printed report. This entry will be used to contain a special code that controls the functions of single, double, or triple spacing, or skipping from one page to the next when printing a line. It should be noted at this point that a single position defined with a FILLER should always be reserved at the beginning of the entries to describe the format of the printed report. This does not cause a space to occur in the output line as do the other filler entries.

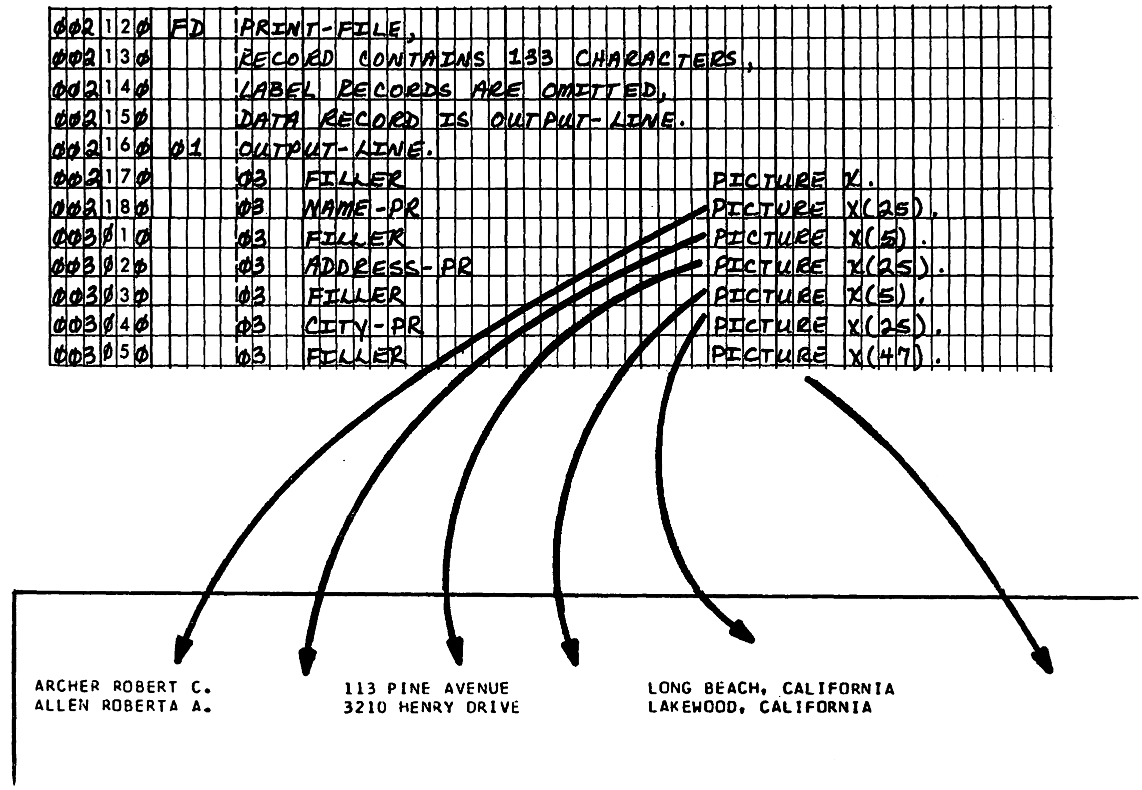
Following the one position FILLER is the 03 Level Indicator with the related entry NAME-PR. This entry describes where the Name field on the card is to be printed on the output line.

The entry following NAME-PR consists of FILLER with a PICTURE X(5). This entry indicates that there are to be five blank spaces following NAME-PR.

The remaining entries describe where the "Address" and "City" are to be printed on the output report.

The following diagram illustrates the entries for "PRINT-FILE" and the relationship of the PICTURE Clause and the printed report.

EXAMPLE



```
002120 FD  PRINT-FILE,
002130     RECORD CONTAINS 133 CHARACTERS,
002140     LABEL RECORDS ARE OMITTED,
002150     DATA RECORD IS OUTPUT-LINE.
002160 01  OUTPUT-LINE.
002170     03  FILLER                      PICTURE X.
002180     03  NAME-PR                     PICTURE X(25).
003010     03  FILLER                      PICTURE X(5).
003020     03  ADDRESS-PR                  PICTURE X(25).
003030     03  FILLER                      PICTURE X(5).
003040     03  CITY-PR                     PICTURE X(25).
003050     03  FILLER                      PICTURE X(47).
```

```
ARCHER ROBERT C.          113 PINE AVENUE          LONG BEACH, CALIFORNIA
ALLEN ROBERTA A.          3210 HENRY DRIVE         LAKEWOOD, CALIFORNIA
```

Figure 2-34 Record Description Entries for Output-Line and Related Output

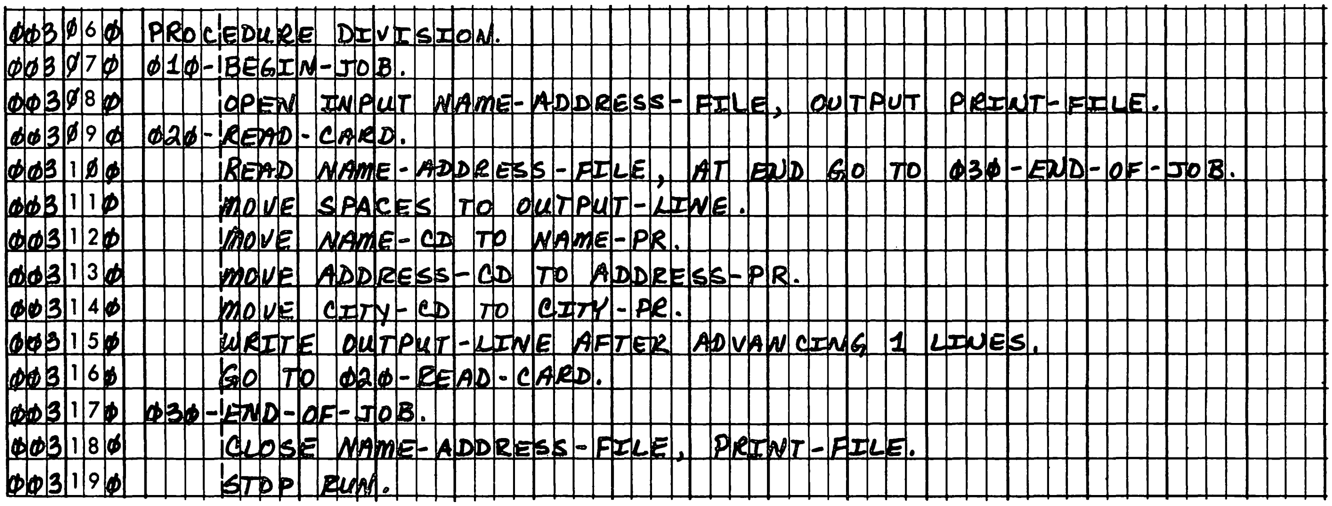
Note from the example in Figure 2-34 that each of the 133 positions which are to be used on the output listing must be specified by Elementary Items within the Group Item OUTPUT-LINE. A Group Item is any item which is further subdivided, such as OUTPUT-LINE. It should be noted also that a group item need not be on the 01 level. It may be on any level so long as it is further subdivided by Elementary Items.

PROCEDURE DIVISION

The Procedure Division of a source program specifies those procedures needed to solve a given problem. These steps (computations, logical decisions, input/output, etc.) are expressed in meaningful statements, similar to English, which employ the concept of verbs to denote actions, statements and sentences to describe procedures. The Procedure Division must begin in Area A with the header "PROCEDURE DIVISION" which is followed by a period. This header must be recorded on a line by itself.

The Procedure Division used in the sample program is illustrated below.

EXAMPLE



```
003060 PROCEDURE DIVISION.
003070 010-BEGIN-JOB.
003080     OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.
003090 020-READ-CARD.
003100     READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.
003110     MOVE SPACES TO OUTPUT-LINE.
003120     MOVE NAME-CD TO NAME-PR.
003130     MOVE ADDRESS-CD TO ADDRESS-PR.
003140     MOVE CITY-CD TO CITY-PR.
003150     WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.
003160     GO TO 020-READ-CARD.
003170 030-END-OF-JOB.
003180     CLOSE NAME-ADDRESS-FILE, PRINT-FILE.
003190     STOP RUN.
```

Figure 2-35 Procedure Division

The basic structure of the Procedure Division consists of PARAGRAPHS containing a series of SENTENCES.

Paragraphs are logical entities consisting of one or more sentences. Each paragraph must begin with a paragraph name which must start in Area A. The sample program consists of three paragraphs, the 010-BEGIN-JOB paragraph, the 020-READ-CARD paragraph, and the 030-END-OF-JOB paragraph. A paragraph ends at the next paragraph name or at the end of the Procedure Division.

It should be noted that the paragraph names are meaningful, that is, they indicate the type of processing which will take place within the paragraph. In addition, there is a prefix number which is placed on each paragraph name. This number is in an ascending sequence for each subsequent paragraph name in the program. Numbering the paragraph names in this manner makes it easier, especially in larger programs, to determine where in the Procedure Division a particular paragraph is located.

A sentence is a single statement or a series of statements terminated by a period and followed by a space. A single comma or semicolon or the word THEN may be used as a separator between statements. Commas, semicolons, and the word THEN have no effect on the program and are used merely to improve readability. The period, on the other hand, is a required punctuation symbol which indicates the end of a sentence.

COBOL verbs are the basis of the Procedure Division of a source program

The basic classification of COBOL verbs are presented in the following list.

Input/Output

- OPEN
- READ
- WRITE
- CLOSE
- ACCEPT
- DISPLAY

Data Manipulation

- MOVE
- EXAMINE

Arithmetic

- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE
- COMPUTE

Sequence Control

- GO TO
- ALTER
- PERFORM
- STOP
- EXIT

Compiler-Directing

- COPY
- ENTER
- NOTE

Table-Handling

- SEARCH
- SET

OPEN Statement

The first paragraph in the Procedure Division contains the Paragraph Name 010-BEGIN-JOB and the OPEN statement.

```
003060 PROCEDURE DIVISION.
003070 010-BEGIN-JOB.
003080     OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.
003090 020-READ-CARD.
```

Figure 2-36 OPEN Statement

The Paragraph Name, also called a Procedure Name, may be composed of not more than 30 characters and may consist of the digits 0-9, the letters of the alphabet A through Z or the hyphen, although a paragraph name may not end or begin with a hyphen. Paragraph Names may be composed solely of numeric characters. No blanks are permitted within the name.

The OPEN statement initiates the processing of files and must be executed prior to any other input/output statement for any file. The OPEN statement, by itself, does not make an input record available for processing; a READ statement must be executed to obtain the first data-record. For an output file, an OPEN statement makes available an area for development of the first output record.

The format of an OPEN statement is:

```
                               ┌ REVERSED         ┐
OPEN [INPUT {file-name         │                  │   }...]
                               └ WITH NO REWIND   ┘

     [OUTPUT {file-name [WITH NO REWIND] }...]

     [I-O  {file-name} ...]
```

Figure 2-37 OPEN Statement – Format Notation

In the OPEN statement format illustrated, it can be seen that the word INPUT must precede the filename of the input file to be opened, and the word OUTPUT must precede the output file to be opened. Thus, in the example in Figure 2-36, the verb OPEN is followed by the word INPUT and then the filename of the card input file, NAME-ADDRESS-FILE, follows. Since there is only one input file, the word OUTPUT is specified next followed by the name of the printer output file, PRINT-FILE. Note that the OPEN verb need only be specified one time in order to open both the card file and the printer file. The OPEN sentence must be terminated by a period.

READ STATEMENT

The next statement in the sample program is the READ statement.

EXAMPLE

```
003070 010-BEGIN-JOB.
003080     OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.
003090 020-READ-CARD.
003100     READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.
```

Figure 2-38 READ Statement

The function of the READ statement is to make available a logical record from an input file and to allow performance of specified operations when end-of-file is detected. End-of-file is detected by the presence of the end-of-data card, which, on the System/360 and System/370, is a card with the values /* in columns 1 and 2.

```
READ file-name  RECORD [INTO identifier]

  { AT END      }
  {             }  imperative-statement
  { INVALID KEY }
```

Figure 2-39 READ Statement — Format Notation

When a Read statement is executed, the next logical record in the named file becomes accessible in the input are defined by the associated Record Description entry. The file-name must be defined by a File Description entry in the Data Division.

The record remains available in the input area until the next READ statement (or a CLOSE statement) for that file is executed.

The AT END option is required for files for which access is sequential. The AT END portion of the READ statement is executed when an end-of-file condition is detected, that is, when the end-of-data card is read.

In the example program, when the end-of-data card is detected, the computer will execute the instructions following the paragraph name 030-END-OF-JOB.

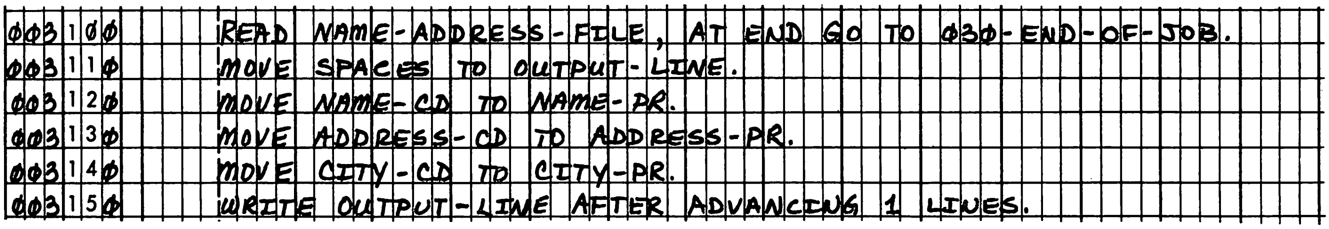
Once the AT END portion of a READ statement has been executed for a file, any subsequent attempt to read from that file or to refer to logical records in that file constitutes an error, unless subsequent CLOSE and OPEN statements have been executed.

MOVE Statement

After a card has been read into storage by means of the READ statement, the next operation to be performed is to clear the printer output area and then move the fields to be printed from the input area to the output area. This is accomplished by means of the MOVE statements.

The following example illustrates the use of the MOVE statements as they are used in the sample problem.

EXAMPLE



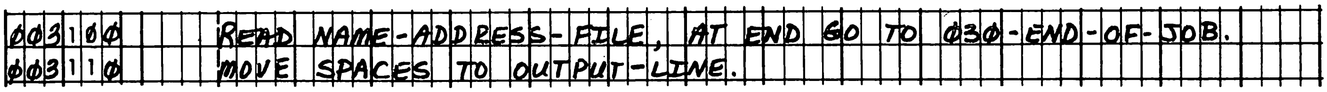
```
003100    READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.
003110    MOVE SPACES TO OUTPUT-LINE.
003120    MOVE NAME-CD TO NAME-PR.
003130    MOVE ADDRESS-CD TO ADDRESS-PR.
003140    MOVE CITY-CD TO CITY-PR.
003150    WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.
```

Figure 2-40 MOVE Statements

The MOVE statement is used to move data from one area of main storage to another.

The MOVE statement on Line 110 illustrates the use of a FIGURATIVE CONSTANT. A Figurative Constant in COBOL is a standard data-name to which a value has been assigned. The Figurative Constant SPACE or SPACES represents one or more "blanks" or "spaces" when specified in a COBOL Program.

EXAMPLE



```
003100    READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.
003110    MOVE SPACES TO OUTPUT-LINE.
```

Figure 2-41 MOVE Statement with Figurative Constant

In the segment of the program illustrated above the effect of the MOVE statement using the Figurative Constant SPACES is to clear the printer output area referenced by OUTPUT-LINE by moving spaces throughout this area. This procedure is necessary to insure that extraneous information is not present in the output area from previous operations.

The MOVE statement has the following format:

```
      ⎧ identifier-1 ⎫
MOVE  ⎨              ⎬  TO  identifier-2  [identifier-3]...
      ⎩ literal      ⎭
```

Figure 2-42 MOVE Statement-Format Notation

From the format notation it can be seen that when the MOVE statement is executed, the information stored in the field specified as "identifier-1" is moved to the area designated by "identifier-2". It should be noted that the word "identifier" when specified in imperative statements such as the MOVE statement refers to the data-name which is used to give a field a symbolic name.

The use of literals in the MOVE statement are explained in Chapter 3. Literals are actual numbers, words, or special characters which may be used in certain COBOL statements.

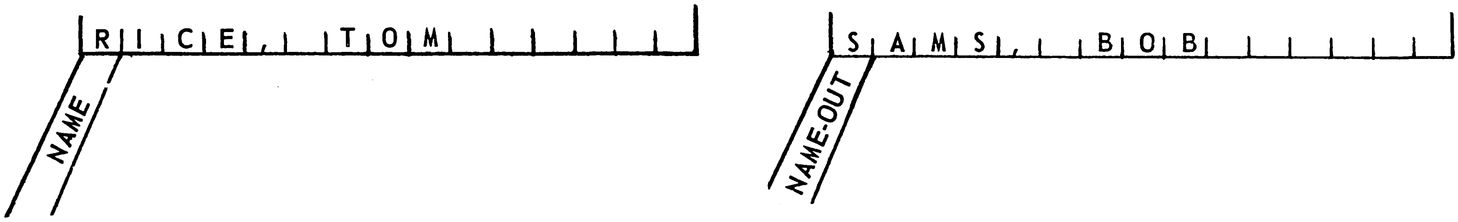
When both the field from which the data is to be moved and the receiving field are alphameric (defined with a Picture X or both fields are Group Items), then the data is placed in the receiving fields aligned to the left.

The following examples illustrate the operation of the MOVE statement when processing alphanumeric fields.

EXAMPLE 1:

```
       03  NAME                          PICTURE X(15).

       03  NAME-OUT                      PICTURE X(15).

       MOVE NAME TO NAME-OUT.
```

Before Execution:



After Execution:

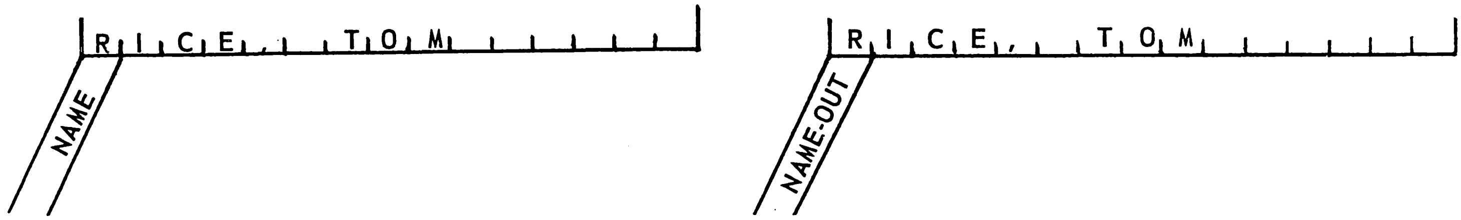


Figure 2-43 Example 1 of MOVE Statement

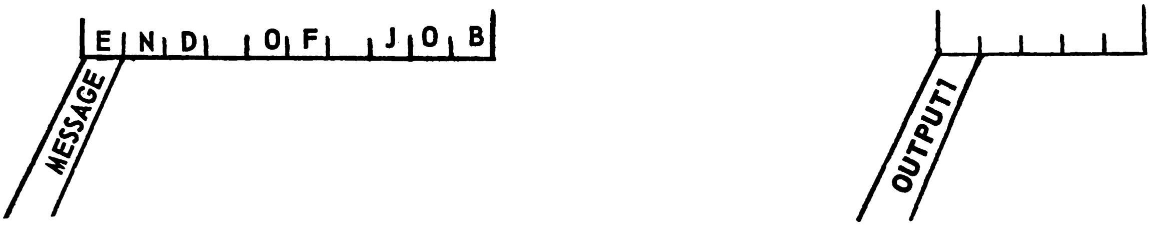
Note that the data in the receiving field is replaced by the data in the sending field and the data in the sending field remains the same.

If the receiving field is smaller than the sending field, "truncation" will take place on the right end of the data when it is placed in the receiving field.

EXAMPLE 2:

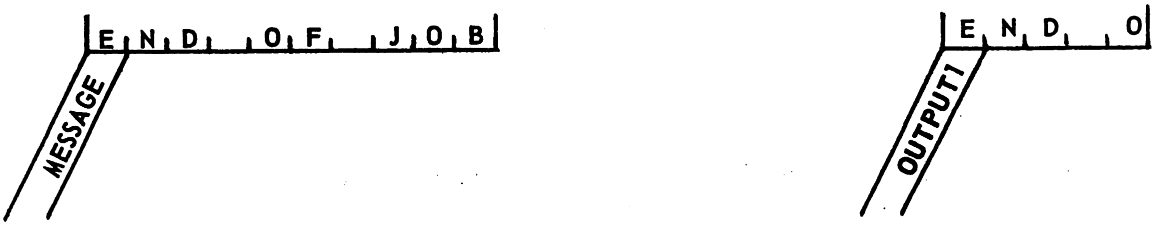
```
       03  MESSAGE                   PICTURE X(10).

       03  OUTPUT1                   PICTURE X(5).

       MOVE MESSAGE TO OUTPUT1.
```

Before Execution:



E N D O F J O B
MESSAGE
OUTPUT1

After Execution:



E N D O F J O B
MESSAGE
E N D O
OUTPUT1

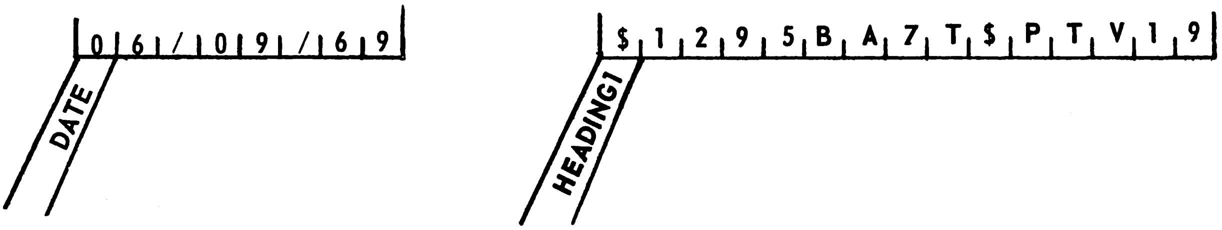
Figure 2-44 Example 2 of MOVE Statement

If the receiving field is larger than the sending field when moving alphameric fields, the right portion of the receiving field will be filled with blanks.

EXAMPLE 3:

```
    03  DATE                          PICTURE X(8).

    03  HEADING1                      PICTURE X(15).

    MOVE DATE TO HEADING1.
```

Before Execution:


0 6 / 0 9 / 6 9
DATE
$ 1 2 9 5 B A 7 T $ P T V 1 9
HEADING1

After Execution:

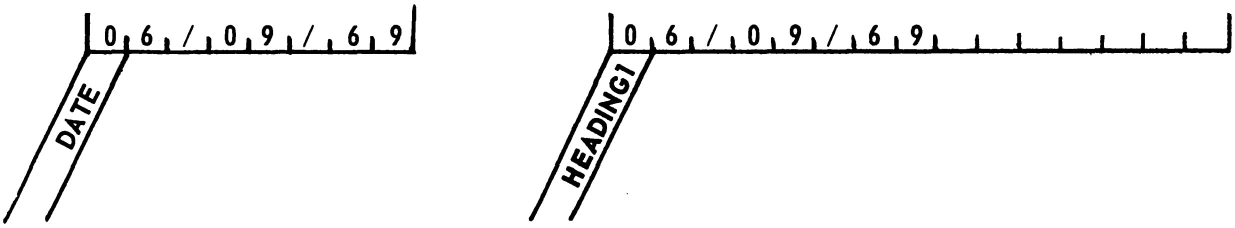

0 6 / 0 9 / 6 9
DATE
0 6 / 0 9 / 6 9
HEADING1

Figure 2-45 Example 3 of MOVE Statement

Normally both the sending and receiving fields will be of equal length. If they are not, a "warning" diagnostic will be printed at compilation time.

WRITE Statement

After the fields have been moved to the output area, the WRITE statement is used to cause a line to be printed.

The Write statement contained in the sample program is illustrated below. Note that the optional entry "FROM identifier-1" is not used in this program.

EXAMPLE

```
003140          MOVE CITY-CD TO CITY-PR.
003150          WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.
003160          GO TO 020-READ-CARD.
```

Figure 2-46 WRITE Statement

The format of the WRITE statement is:

```
WRITE record-name [FROM identifier-1]

┌ ( BEFORE )              ( identifier-2 LINES ) ┐
│ {        }  ADVANCING   { integer LINES      } │
└ ( AFTER  )              ( mnemonic-name      ) ┘
```

Figure 2-47 WRITE Statement – Format Notation

The AFTER ADVANCING option is used for output destined to be printed or punched. When this option is used, the first character in each logical record for the file must be reserved for the control character. This is done in the sample program in the Data Division by the FILLER with a PICTURE X as the first entry when describing OUTPUT-LINE. If a WRITE statement with an ADVANCING option is written for a record in a file, every WRITE statement for records in the same file must contain an ADVANCING option. When the AFTER ADVANCING option is used, "integer" must be non-negative and have a value of less than 100. In the example in Figure 2-46 it can be seen that the integer value "1" is used to indicate single spacing is to occur on the report. The value 2 designates double spacing and the value 3, triple spacing.

An OPEN statement must be executed prior to executing the first WRITE statement for a file. After the WRITE statement is executed, the logical record named by "record-name" is no longer available.

GO TO Statement

After a card has been read, the fields are moved to the output print line, a line is printed, and the sequence of operations is to be repeated. This is accomplished by the GO TO statement.

EXAMPLE

```
003150    WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.
003160    GO TO 020-READ-CARD.
```

Figure 2-48 GO TO Statement

The GO TO statement transfers control from one portion of the program to another and has the following format:

```
GO TO procedure-name-1
```

Figure 2-49 GO TO Statement – Format Notation

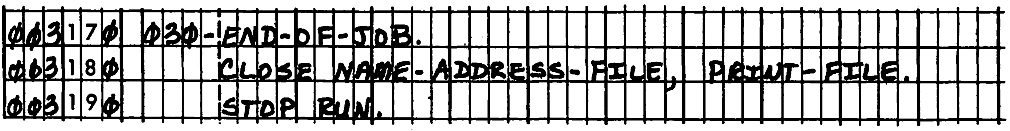
The GO TO statement provides a means of transferring the path of a program to a designated paragraph or section. In the example in Figure 2-48, the GO TO 020-READ-CARD statement will cause control to be passed to the statement immediately following the paragraph-name 020-READ-CARD. Thus, after the GO TO statement is executed, another card will be read by the READ statement which immediately follows the paragraph-name.

Note also the use of the GO TO statement in the READ statement which executed "end-of-file" operations when the end-of-data card is read.

CLOSE Statement

In the sample program when the "/*" card is detected indicating that all data has been read and processed, the program branches to the 030-END-OF-JOB Routine in which the files are closed and the program is completed.

EXAMPLE



```
003170 030-END-OF-JOB.
003180     CLOSE NAME-ADDRESS-FILE, PRINT-FILE.
003190     STOP RUN.
```

Figure 2-50 CLOSE Statement

The CLOSE statement is used to terminate the processing of one or more units or files. The format of the CLOSE statement is:

```
                              ┌ REEL ┐        ( NO REWIND )
CLOSE    file-name-1          │      │ [WITH  {           }  ]
                              └ UNIT ┘        ( LOCK      )

                            ┌ REEL ┐        ( NO REWIND )
         [file-name-2       │      │ [WITH  {           }  ] ] ...
                            └ UNIT ┘        ( LOCK      )
```

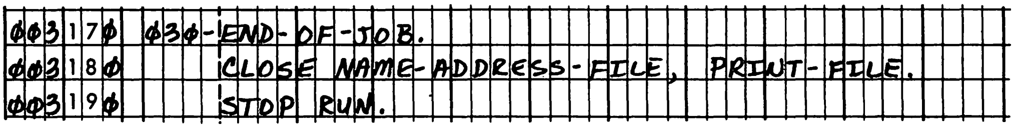
Figure 2-51 CLOSE Statement – Format Notation

When a CLOSE statement is specified, closing procedures are executed for the current unit of the file. The CLOSE statement may only be specified for a file that is open. After a CLOSE statement has been executed for a file, an OPEN statement must be executed before any other reference can be made to that file.

STOP Statement

The last statement in the sample program is the STOP statement.

EXAMPLE



```
003170 030-END-OF-JOB.
003180     CLOSE NAME-ADDRESS-FILE, PRINT-FILE.
003190     STOP RUN.
```

$$\underline{\text{STOP}} \quad \begin{Bmatrix} \underline{\text{RUN}} \\ \text{literal} \end{Bmatrix}$$

Figure 2-52 STOP Statement

The STOP RUN statement terminates execution of the object program and returns control to the "operating system".

This chapter has been designed to introduce the basic concepts of computer programming using COBOL. Subsequent chapters of the text introduce additional COBOL verbs and programming techniques.

SAMPLE PROBLEM

To illustrate the basic concepts of programming and the entries required to write a COBOL program, a sample problem is introduced in this chapter in which a file of Name and Address cards is used to prepare a Name and Address Listing.

The complete documentation for this problem, including the input, output, flowchart, and COBOL program are illustrated on the following pages.

SAMPLE PROBLEM

Input: Name and Address Cards

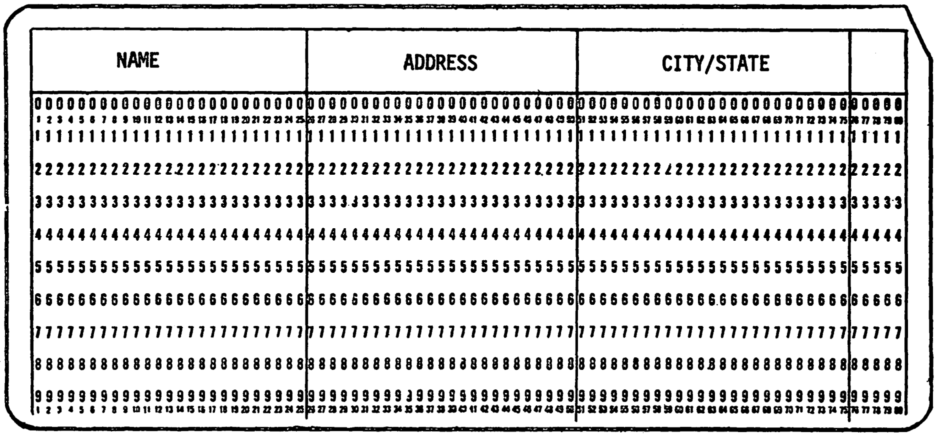
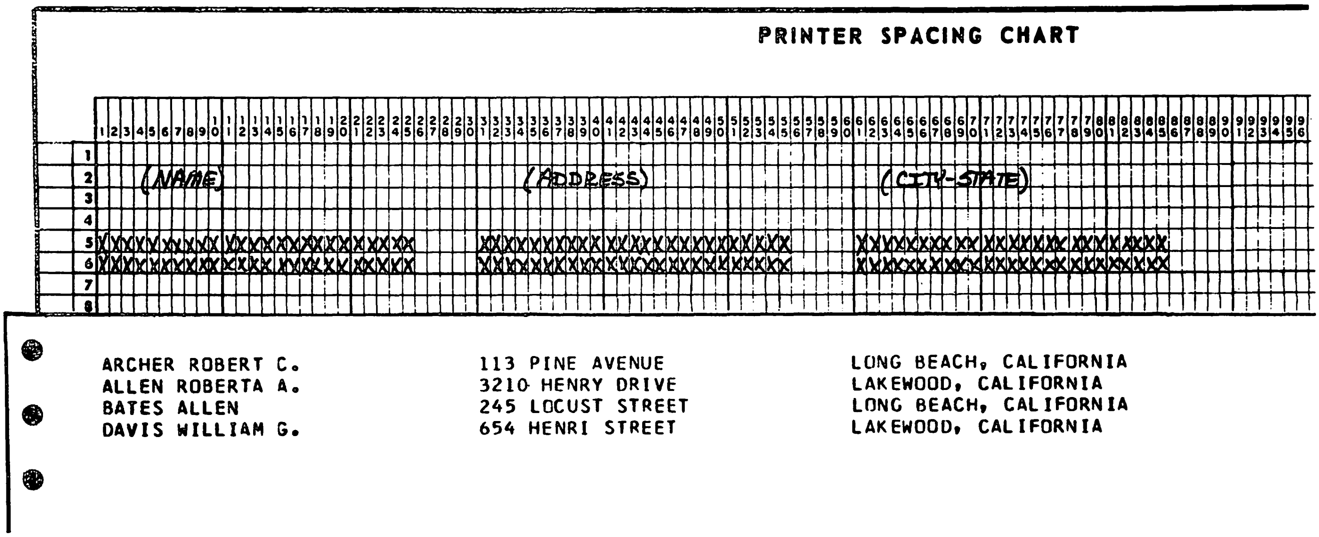

NAME
ADDRESS
CITY/STATE

Figure 2-53 Input

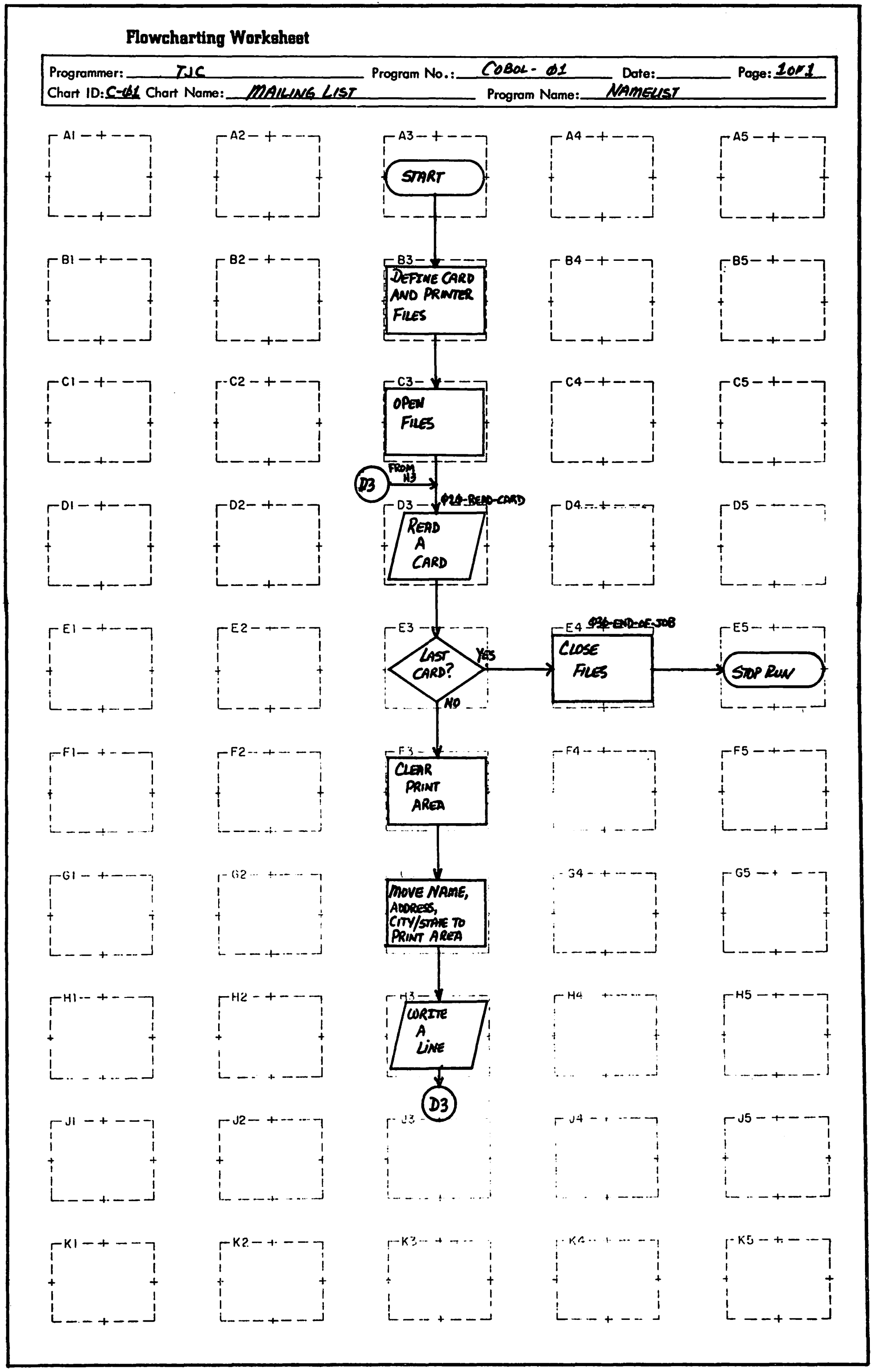
Output: Printer Report


PRINTER SPACING CHART
(NAME)
(ADDRESS)
(CITY-STATE)

```
ARCHER ROBERT C.              113 PINE AVENUE               LONG BEACH, CALIFORNIA
ALLEN ROBERTA A.              3210 HENRY DRIVE              LAKEWOOD, CALIFORNIA
BATES ALLEN                   245 LOCUST STREET             LONG BEACH, CALIFORNIA
DAVIS WILLIAM G.              654 HENRI STREET              LAKEWOOD, CALIFORNIA
```

Figure 2-54 Output from the Sample Program

Flowchart



Flowcharting Worksheet

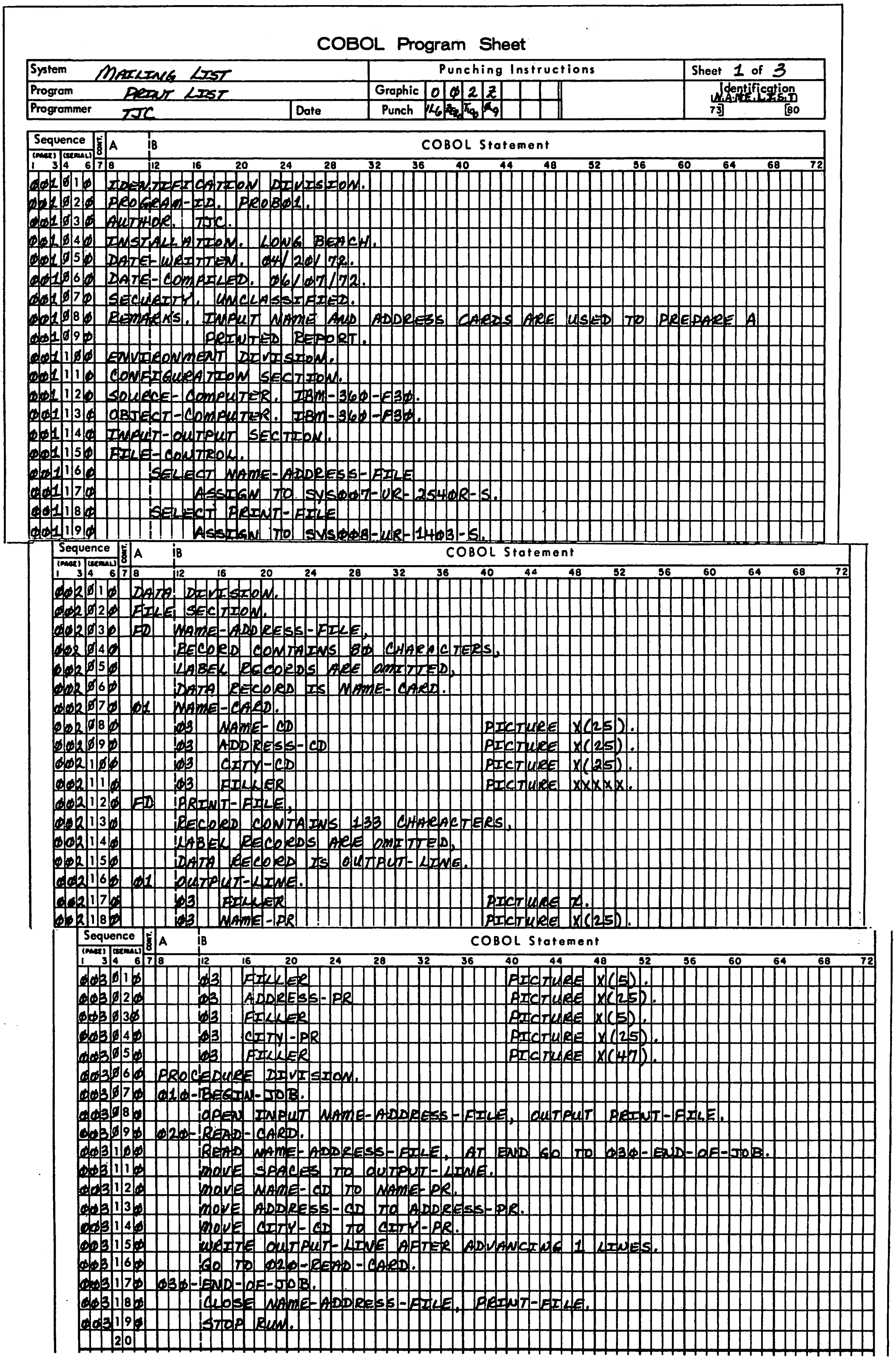
Programmer: TJC Program No.: COBOL-01 Date: Page: 1 OF 1
Chart ID: C-01 Chart Name: MAILING LIST Program Name: NAMELIST

START
DEFINE CARD AND PRINTER FILES
OPEN FILES
D3
FROM H3
020-READ-CARD
READ A CARD
LAST CARD?
YES
NO
030-END-OF-JOB
CLOSE FILES
STOP RUN
CLEAR PRINT AREA
MOVE NAME, ADDRESS, CITY/STATE TO PRINT AREA
WRITE A LINE
D3
A1
A2
A3
A4
A5
B1
B2
B3
B4
B5
C1
C2
C3
C4
C5
D1
D2
D3
D4
D5
E1
E2
E3
E4
E5
F1
F2
F3
F4
F5
G1
G2
G4
G5
H1
H2
H3
H4
H5
J1
J2
J3
J4
J5
K1
K2
K3
K4
K5

Figure 2-55 Flowchart for the Sample Problem

Program

The coding of the sample program is illustrated below.



COBOL Program Sheet

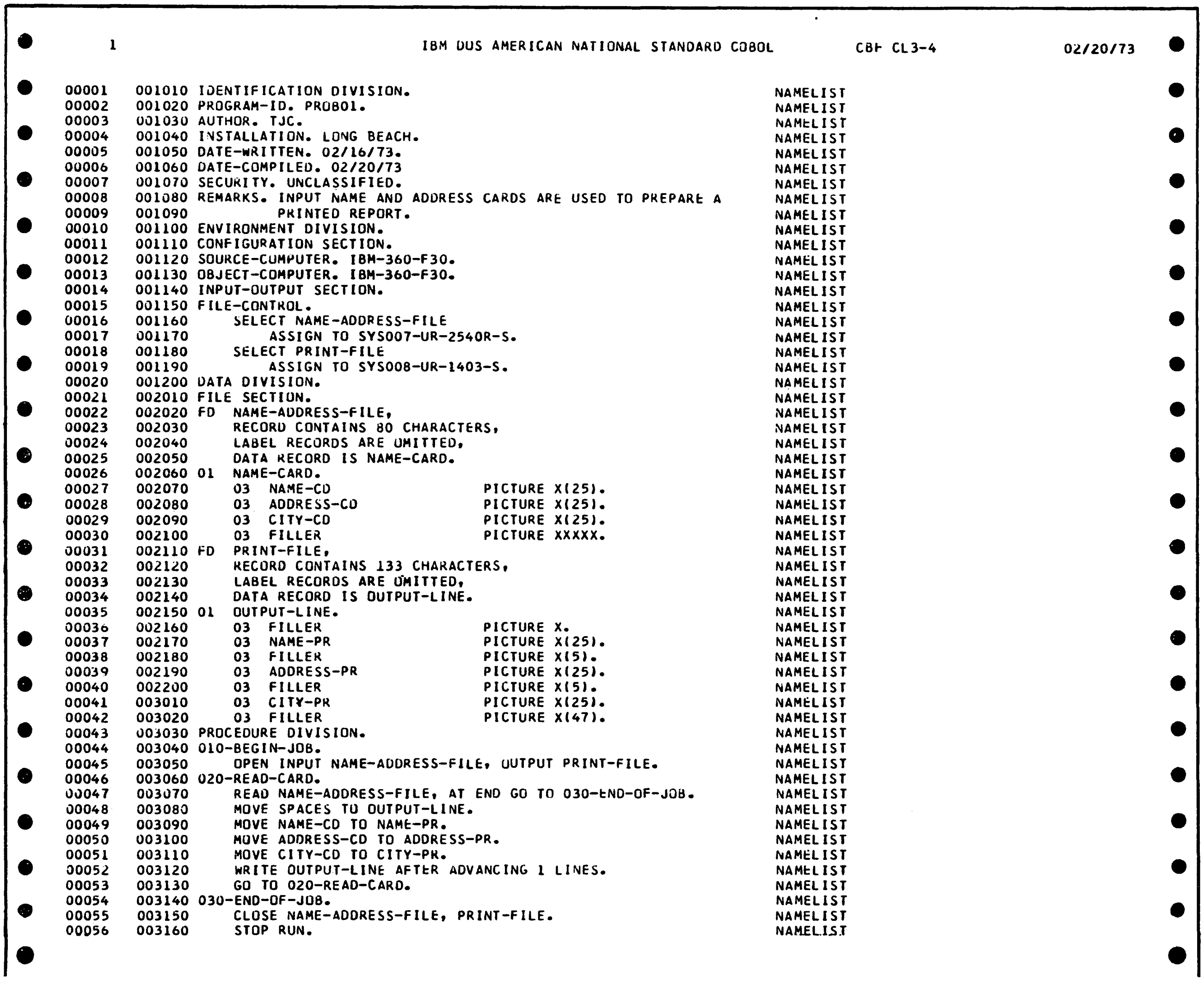
System MAILING LIST
Program PRINT LIST
Programmer TJC
Date
Punching Instructions
Graphic 0 0 2 Z
Punch [illegible]
Sheet 1 of 3
Identification NAMELIST 73 80

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. PROB01.
001030 AUTHOR. TJC.
001040 INSTALLATION. LONG BEACH.
001050 DATE-WRITTEN. 04/20/72.
001060 DATE-COMPILED. 06/07/72.
001070 SECURITY. UNCLASSIFIED.
001080 REMARKS. INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A
001090         PRINTED REPORT.
001100 ENVIRONMENT DIVISION.
001110 CONFIGURATION SECTION.
001120 SOURCE-COMPUTER. IBM-360-F30.
001130 OBJECT-COMPUTER. IBM-360-F30.
001140 INPUT-OUTPUT SECTION.
001150 FILE-CONTROL.
001160     SELECT NAME-ADDRESS-FILE
001170         ASSIGN TO SYS007-UR-2540R-S.
001180     SELECT PRINT-FILE
001190         ASSIGN TO SYS008-UR-1403-S.
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  NAME-ADDRESS-FILE,
002040     RECORD CONTAINS 80 CHARACTERS,
002050     LABEL RECORDS ARE OMITTED,
002060     DATA RECORD IS NAME-CARD.
002070 01  NAME-CARD.
002080     03  NAME-CD                 PICTURE X(25).
002090     03  ADDRESS-CD              PICTURE X(25).
002100     03  CITY-CD                 PICTURE X(25).
002110     03  FILLER                  PICTURE XXXXX.
002120 FD  PRINT-FILE,
002130     RECORD CONTAINS 133 CHARACTERS,
002140     LABEL RECORDS ARE OMITTED,
002150     DATA RECORD IS OUTPUT-LINE.
002160 01  OUTPUT-LINE.
002170     03  FILLER                  PICTURE X.
002180     03  NAME-PR                 PICTURE X(25).
003010     03  FILLER                  PICTURE X(5).
003020     03  ADDRESS-PR              PICTURE X(25).
003030     03  FILLER                  PICTURE X(5).
003040     03  CITY-PR                 PICTURE X(25).
003050     03  FILLER                  PICTURE X(47).
003060 PROCEDURE DIVISION.
003070 010-BEGIN-JOB.
003080     OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.
003090 020-READ-CARD.
003100     READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.
003110     MOVE SPACES TO OUTPUT-LINE.
003120     MOVE NAME-CD TO NAME-PR.
003130     MOVE ADDRESS-CD TO ADDRESS-PR.
003140     MOVE CITY-CD TO CITY-PR.
003150     WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.
003160     GO TO 020-READ-CARD.
003170 030-END-OF-JOB.
003180     CLOSE NAME-ADDRESS-FILE, PRINT-FILE.
003190     STOP RUN.
   20
```

Figure 2-56 Coding of Sample Program

Source Listing

The source listing from the computer of the sample program is illustrated below.



```
     1                                   IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4               02/20/73

 00001   001010 IDENTIFICATION DIVISION.                                         NAMELIST
 00002   001020 PROGRAM-ID. PROB01.                                              NAMELIST
 00003   001030 AUTHOR. TJC.                                                     NAMELIST
 00004   001040 INSTALLATION. LONG BEACH.                                        NAMELIST
 00005   001050 DATE-WRITTEN. 02/16/73.                                          NAMELIST
 00006   001060 DATE-COMPILED. 02/20/73                                          NAMELIST
 00007   001070 SECURITY. UNCLASSIFIED.                                          NAMELIST
 00008   001080 REMARKS. INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A      NAMELIST
 00009   001090          PRINTED REPORT.                                         NAMELIST
 00010   001100 ENVIRONMENT DIVISION.                                            NAMELIST
 00011   001110 CONFIGURATION SECTION.                                           NAMELIST
 00012   001120 SOURCE-COMPUTER. IBM-360-F30.                                    NAMELIST
 00013   001130 OBJECT-COMPUTER. IBM-360-F30.                                    NAMELIST
 00014   001140 INPUT-OUTPUT SECTION.                                            NAMELIST
 00015   001150 FILE-CONTROL.                                                    NAMELIST
 00016   001160     SELECT NAME-ADDRESS-FILE                                     NAMELIST
 00017   001170         ASSIGN TO SYS007-UR-2540R-S.                             NAMELIST
 00018   001180     SELECT PRINT-FILE                                            NAMELIST
 00019   001190         ASSIGN TO SYS008-UR-1403-S.                              NAMELIST
 00020   001200 DATA DIVISION.                                                   NAMELIST
 00021   002010 FILE SECTION.                                                    NAMELIST
 00022   002020 FD  NAME-ADDRESS-FILE,                                           NAMELIST
 00023   002030     RECORD CONTAINS 80 CHARACTERS,                               NAMELIST
 00024   002040     LABEL RECORDS ARE OMITTED,                                   NAMELIST
 00025   002050     DATA RECORD IS NAME-CARD.                                    NAMELIST
 00026   002060 01  NAME-CARD.                                                   NAMELIST
 00027   002070     03  NAME-CD                 PICTURE X(25).                   NAMELIST
 00028   002080     03  ADDRESS-CD              PICTURE X(25).                   NAMELIST
 00029   002090     03  CITY-CD                 PICTURE X(25).                   NAMELIST
 00030   002100     03  FILLER                  PICTURE XXXXX.                   NAMELIST
 00031   002110 FD  PRINT-FILE,                                                  NAMELIST
 00032   002120     RECORD CONTAINS 133 CHARACTERS,                              NAMELIST
 00033   002130     LABEL RECORDS ARE OMITTED,                                   NAMELIST
 00034   002140     DATA RECORD IS OUTPUT-LINE.                                  NAMELIST
 00035   002150 01  OUTPUT-LINE.                                                 NAMELIST
 00036   002160     03  FILLER                  PICTURE X.                       NAMELIST
 00037   002170     03  NAME-PR                 PICTURE X(25).                   NAMELIST
 00038   002180     03  FILLER                  PICTURE X(5).                    NAMELIST
 00039   002190     03  ADDRESS-PR              PICTURE X(25).                   NAMELIST
 00040   002200     03  FILLER                  PICTURE X(5).                    NAMELIST
 00041   003010     03  CITY-PR                 PICTURE X(25).                   NAMELIST
 00042   003020     03  FILLER                  PICTURE X(47).                   NAMELIST
 00043   003030 PROCEDURE DIVISION.                                              NAMELIST
 00044   003040 010-BEGIN-JOB.                                                   NAMELIST
 00045   003050     OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.             NAMELIST
 00046   003060 020-READ-CARD.                                                   NAMELIST
 00047   003070     READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.         NAMELIST
 00048   003080     MOVE SPACES TO OUTPUT-LINE.                                  NAMELIST
 00049   003090     MOVE NAME-CD TO NAME-PR.                                     NAMELIST
 00050   003100     MOVE ADDRESS-CD TO ADDRESS-PR.                               NAMELIST
 00051   003110     MOVE CITY-CD TO CITY-PR.                                     NAMELIST
 00052   003120     WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.                   NAMELIST
 00053   003130     GO TO 020-READ-CARD.                                         NAMELIST
 00054   003140 030-END-OF-JOB.                                                  NAMELIST
 00055   003150     CLOSE NAME-ADDRESS-FILE, PRINT-FILE.                         NAMELIST
 00056   003160     STOP RUN.                                                    NAMELIST
```

Figure 2-57 Source Listing

CHAPTER 2

PROGRAMMING ASSIGNMENT 1

INPUT/OUTPUT OPERATIONS

The following programming assignment should be coded, keypunched, and run. Test data may be found in the appendices.

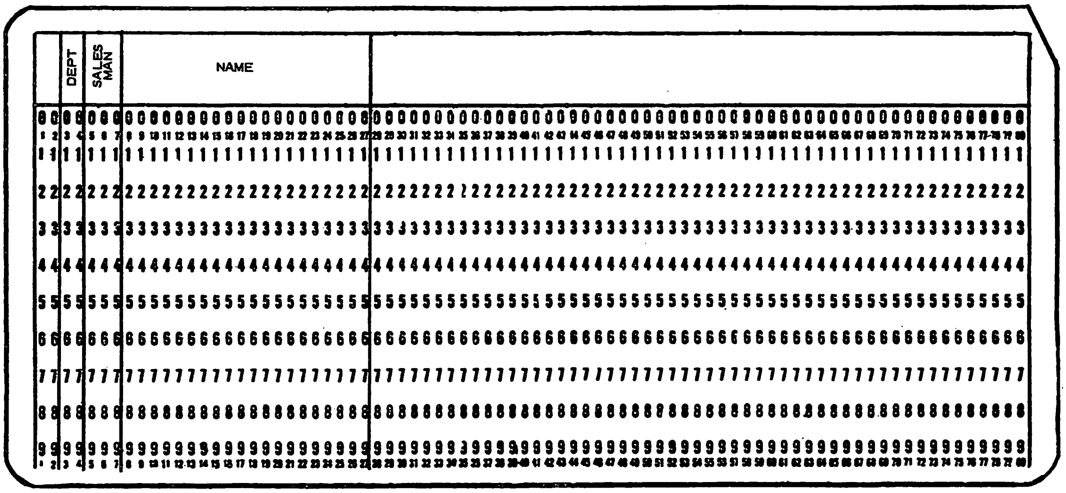
INSTRUCTIONS:

From the Sales Cards illustrated below, a report is to be prepared listing the Salesman Number, the Salesman Name, and the Department Number from each card.

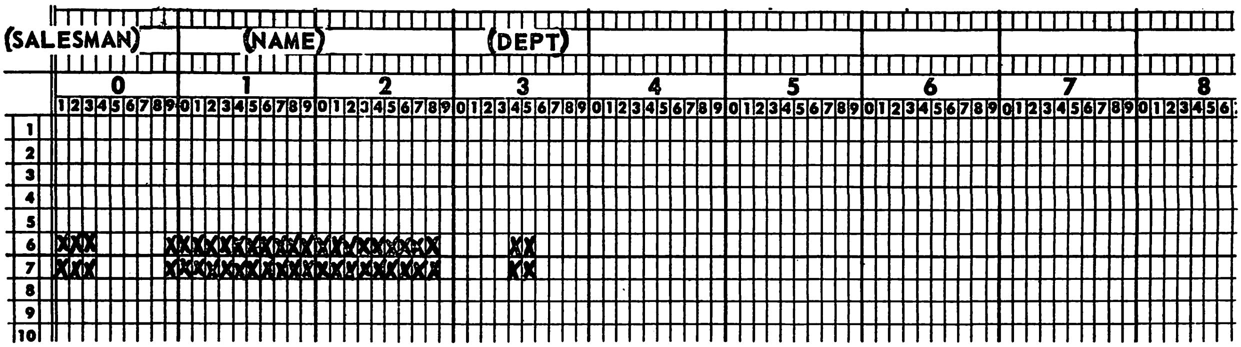
The cards are to be sorted by Salesman Number.

Prepare a Flowchart, and the COBOL Program to produce the required report.

INPUT: Sales Cards



OUTPUT: Salesman Name List



CHAPTER 2

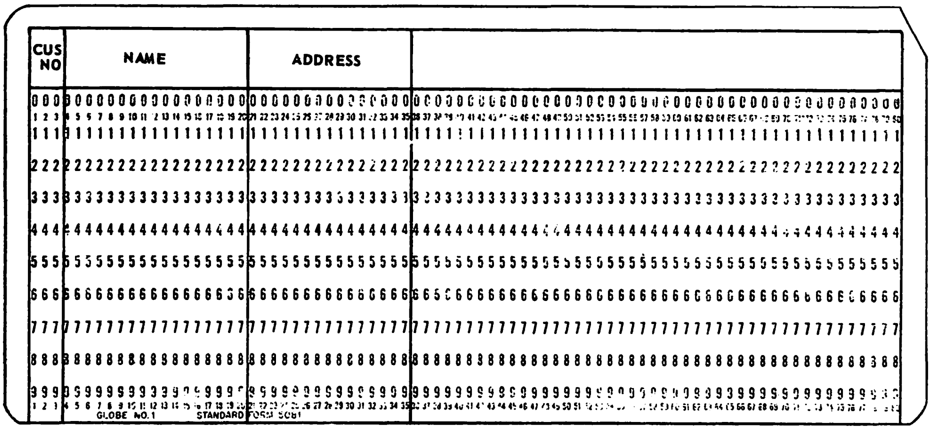
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS:

Write the COBOL program to produce two listings of the customers of a company.

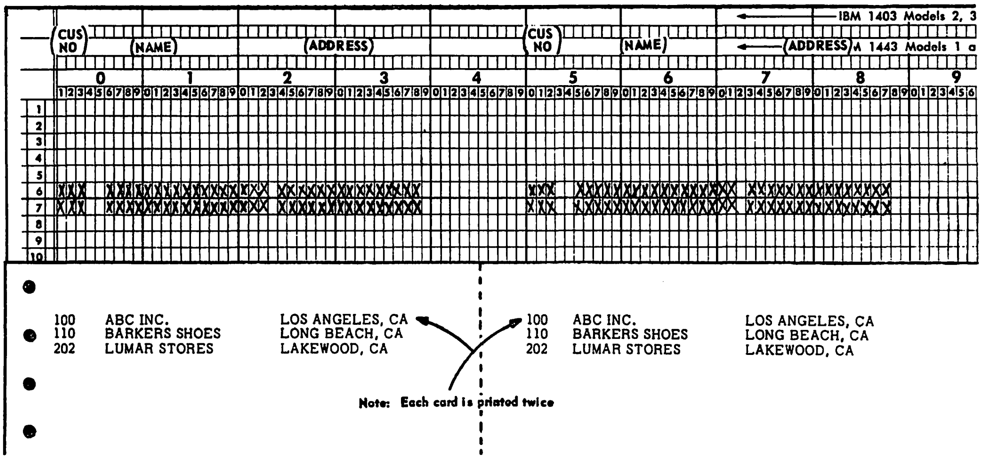
INPUT: Customer Name and Address Cards

Input is to consist of Customer Name and Address Cards that contain the Customer Number, Customer Name, and Customer Address. The format of the cards is illustrated below.


CUS NO
NAME
ADDRESS
GLOBE NO. 1
STANDARD FORM 5081

OUTPUT: Customer Listing

Output is to consist of two listings of the Customers. To reduce computer run time it has been decided to print each of the fields on the card twice on the same line. A Printer Spacing Chart and a segment of the report that is to be prepared is illustrated below.


IBM 1403 Models 2, 3
A 1443 Models 1 a
CUS NO
NAME
ADDRESS
CUS NO
NAME
ADDRESS

```
100  ABC INC.         LOS ANGELES, CA    100  ABC INC.         LOS ANGELES, CA
110  BARKERS SHOES    LONG BEACH, CA     110  BARKERS SHOES    LONG BEACH, CA
202  LUMAR STORES     LAKEWOOD, CA       202  LUMAR STORES     LAKEWOOD, CA
```

Note: Each card is printed twice

TEST DATA – ASSIGNMENT NO. 2

Customer Number (Col 1-3)	Customer Name (Col 4-20)	Customer Address (Col 21-35)
100	ABC Inc.	Los Angeles, CA
110	Barkers Shoes	Long Beach, CA
202	Lumar Stores	Lakewood, CA
278	Jamieson Furn.	San Pedro, CA
442	Nancy's Clothes	Hollywood, CA
668	Julie's Jewelry	Costa Mesa, CA

CHAPTER 3

ADDITION - SUBTRACTION

The arithmetic operations of addition and subtraction are basic to many business applications. Addition in COBOL is accomplished by use of the ADD verb. Subtraction is accomplished by the SUBTRACT verb.

The sample program developed in this chapter illustrates a basic crossfooting operation in which two fields are added together giving a total. The COBOL program is designed to add a Regular Earnings field to an Overtime Earnings field to determine an Employee's Total Earnings.

INPUT

Input for the sample problem consists of a file of Employee Earnings cards. The card format is illustrated below.

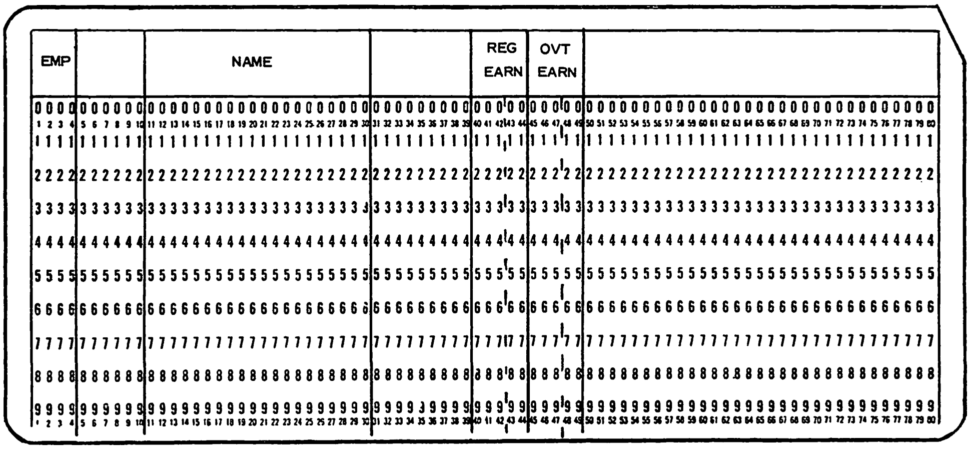
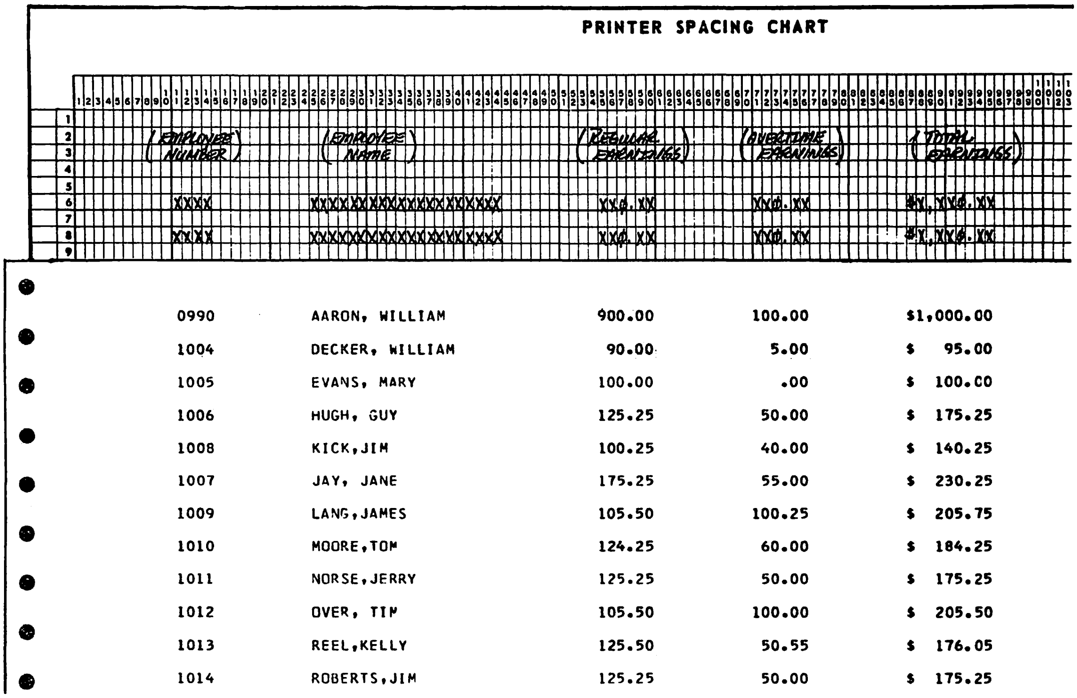


Figure 3-1 Employee Earnings Card

As can be seen from Figure 3-1, the input card contains the Employee Number (col 1-4), the Employee Name (col 11-30), the Regular Earnings (col 40-44) and the Overtime Earnings (col 45-49).

OUTPUT

Output is to consist of a printed report listing the Employee Number, the Employee Name, Regular Earnings, Overtime Earnings, and Total Earnings. An example of the Printer Spacing Chart and the output to be produced is illustrated in Figure 3-2.


PRINTER SPACING CHART
(EMPLOYEE NUMBER)
(EMPLOYEE NAME)
(REGULAR EARNINGS)
(OVERTIME EARNINGS)
(TOTAL EARNINGS)
XXXX
XXXXXXXXXXXXXXXXXXXX
XXØ.XX
XXØ.XX
$X,XXØ.XX
XXXX
XXXXXXXXXXXXXXXXXXXX
XXØ.XX
XXØ.XX
$X,XXØ.XX

```
0990        AARON, WILLIAM               900.00        100.00      $1,000.00

1004        DECKER, WILLIAM               90.00          5.00      $   95.00

1005        EVANS, MARY                  100.00           .00      $  100.00

1006        HUGH, GUY                    125.25         50.00      $  175.25

1008        KICK,JIM                     100.25         40.00      $  140.25

1007        JAY, JANE                    175.25         55.00      $  230.25

1009        LANG,JAMES                   105.50        100.25      $  205.75

1010        MOORE,TOM                    124.25         60.00      $  184.25

1011        NORSE,JERRY                  125.25         50.00      $  175.25

1012        OVER, TIM                    105.50        100.00      $  205.50

1013        REEL,KELLY                   125.50         50.55      $  176.05

1014        ROBERTS,JIM                  125.25         50.00      $  175.25
```

Figure 3-2 Printer Spacing Chart and Output

Note in the segment of the sample report that the Regular Earnings field, Overtime Earnings field, and Total Earnings are to be "zero suppressed" and "edited". Zero suppression refers to the process of suppressing the printing of zeros to the left of the first significant digit of a field. For example, a field consisting of the numbers 0010000 would be printed as "10000" when zero suppressed. "Editing" refers to the process of inserting special characters such as the decimal point, comma, and the dollar sign in fields which are printed.

In the sample output in Figure 3-2 it can be seen that Regular Earnings field and Overtime Earnings field are zero suppressed and edited with a decimal point. The Total Earnings field is zero suppressed and edited with a dollar sign, comma, and decimal point.

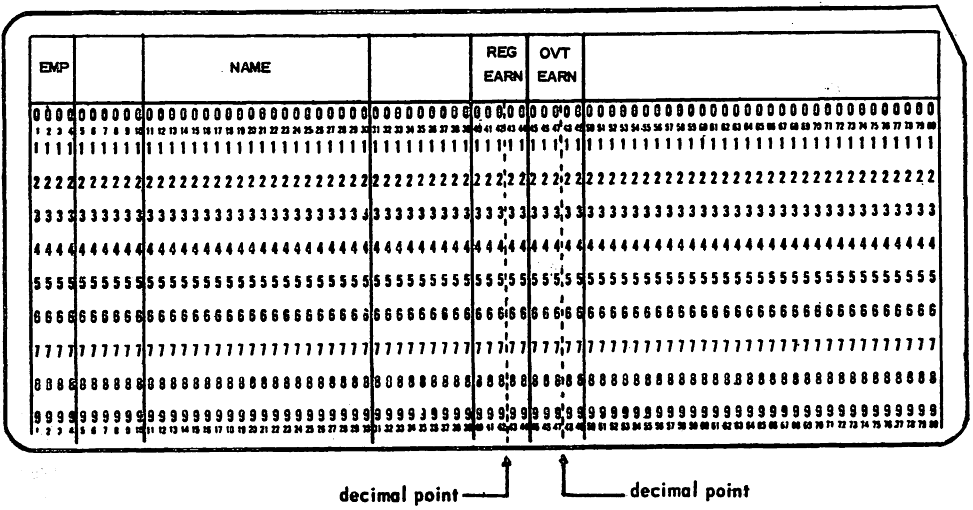
DEFINITION OF NUMERIC DATA

In Chapter 2, the data which was defined for the input card was specified as alphameric data, that is, the PICTURE X clause was used. When data is to be used in arithmetic operations, such as addition or subtraction, or is to be edited on a report, the data must be defined as NUMERIC Data. In order to define data as numeric, the PICTURE 9 clause must be used, where the "9" indicates that the data to be stored in the field will be numeric.

In addition, when fields represent amounts expressed in terms of dollars and cents, the compiler must be informed of the effective location of the decimal points in order to perform the addition operation properly. In most applications, the decimal points are not actually punched on a card. Therefore, the COBOL compiler allows the programmer to specify where the implied decimal point is located through the use of the character "V" within a Picture clause.

In the sample program developed at the end of the chapter, input is to consist of Employee Earnings Cards containing a Regular Earnings field and an Overtime Earnings field. These fields are recorded on the cards in terms of dollars and cents. Thus, the assumed decimal point is two positions to the left of the units position of each field. The following is an illustration of the format of the input card, and the corresponding entries in the File Section to define the card input file.

EXAMPLE



```
002010 FILE SECTION.
002020 FD  TIME-CARD-FILE,
002030     RECORD CONTAINS 80 CHARACTERS,
002040     LABEL RECORDS ARE OMITTED,
002050     DATA RECORD IS CARDIN.
002060 01  CARDIN.
002070     03  EMPLOYEE-CD             PICTURE XXXX.
002080     03  FILLER                  PICTURE X(6).
002090     03  NAME-CD                 PICTURE X(20).
002100     03  FILLER                  PICTURE X(9).
002110     03  REGULAR-EARNINGS-CD     PICTURE 999V99.
002120     03  OVERTIME-EARNINGS-CD    PICTURE 999V99.
002130     03  FILLER                  PICTURE X(31).
```

Alphameric Field

Numeric Fields

Figure 3-3 Definition of Numeric Fields

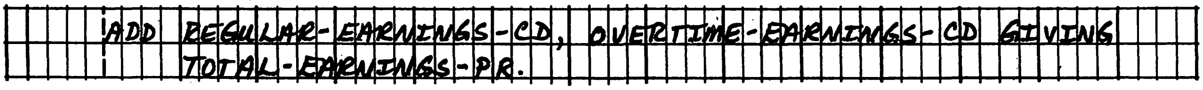
Note in Figure 3-3 that the Picture for the REGULAR-EARNINGS-CD field is expressed as "PICTURE 999V99" and the Picture for the OVERTIME-EARNINGS-CD field is expressed as "PICTURE 999V99". The "9" in the picture indicates that the fields are to be treated as Numeric fields and that they may be involved in arithmetic operations and editing. The "V" in the Picture specifies the position of the assumed decimal point. It should again be noted that storage is not reserved for the "V" and that the "V" is merely used to provide the compiler with the information concerning the decimal alignment of numeric items involved in computations.

Note also from Figure 3-3 that the FILLER field defined on coding line 100 contains the picture entry PICTURE X(9). It should again be noted that the value within the parentheses indicates the number of characters in the field being defined. Thus, this picture clause states that there will be nine alphameric characters in the FILLER field. The "9" within the parentheses has no relationship to the nine which specifies a numeric field for the Regular Earnings and Overtime Earnings field. Thus, the Picture clause PICTURE 9(9) would state that there are nine numeric digits in the field being defined.

ADD STATEMENT

The ADD statement as used in the sample problem to accumulate a total earnings is written as follows:

EXAMPLE



```
ADD REGULAR-EARNINGS-CD, OVERTIME-EARNINGS-CD GIVING
    TOTAL-EARNINGS-PR.
```

Figure 3-4 Sample ADD Statement

When this statement is executed, the field referenced by REGULAR-EARNINGS-CD is added to the field referenced by OVERTIME-EARNINGS-CD and the answer is stored in the area referenced by TOTAL-EARNINGS-PR.

In the example of the ADD statement note the use of the comma between REGULAR-EARNINGS-CD and OVERTIME-EARNINGS-CD. A comma may be used as a "separator" between successive operands of a statement for clarity and ease of reading if desired.

ADD Statement – Format Notation

The COBOL format notation for the ADD statement used in the sample program is illustrated below.

```
    { identifier-1 }  { identifier-2 }  [ identifier-3 ]
ADD {              }  {              }  [              ]   ...
    { literal-1    }  { literal-2    }  [ literal-3    ]
   GIVING identifier-m [ROUNDED] [ON SIZE ERROR imperative-statement]
```

Figure 3-5 ADD Statement – Format Notation

If the GIVING option is used in the ADD statement, the value of identifier-m that follows the word GIVING is made equal to the calculated result.

The ellipsis (. . .) in format notation indicates that the immediately preceding units may occur once, or any number of times in succession. A unit means either a single lower-case word, or a group of lower-case words or one or more reserved words enclosed in brackets or braces.

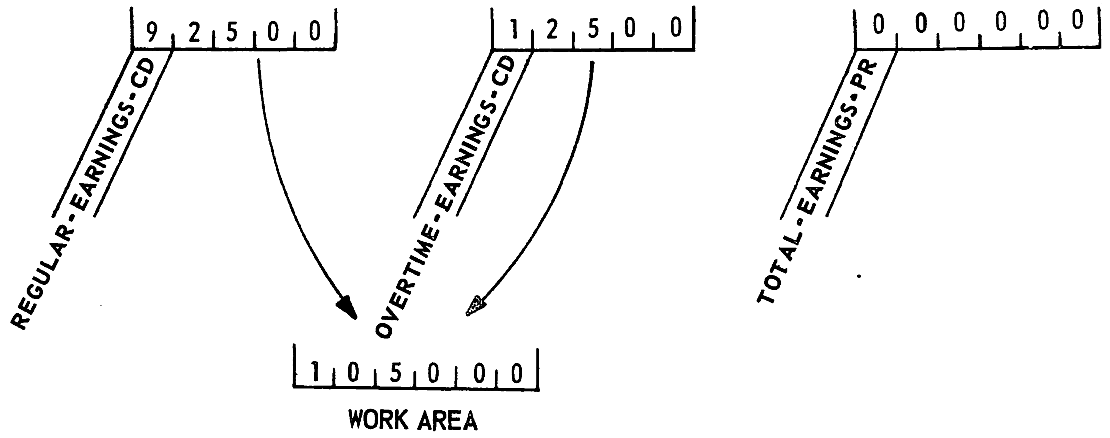
Thus, in the ADD statement an identifier or a numeric literal may be used a number of times. Numeric Literals are merely numbers that may be used within arithmetic statements rather than identifiers which reference fields containing numeric values.

It should be noted that when arithmetic operations take place using the GIVING option the answer is first developed in a work area set up by the compiler and then moved to the area referenced by identifier-m following the word GIVING. The following example illustrates a more detailed analysis of the operation that takes place internally in the computer with the ADD statement when the GIVING option is used.

EXAMPLE

```
    ADD REGULAR-EARNINGS-CD, OVERTIME-EARNINGS-CD GIVING
        TOTAL-EARNINGS-PR.
```

STEP 1: When the ADD statement is executed, the two fields are added and the answer is developed in a "work area" by the compiler.


9 2 5 0 0
REGULAR-EARNINGS-CD
1 2 5 0 0
OVERTIME-EARNINGS-CD
0 0 0 0 0 0
TOTAL-EARNINGS-PR
1 0 5 0 0 0
WORK AREA

STEP 2: Answer developed in the "work area" is moved to the area referenced by the Identifier following the GIVING.

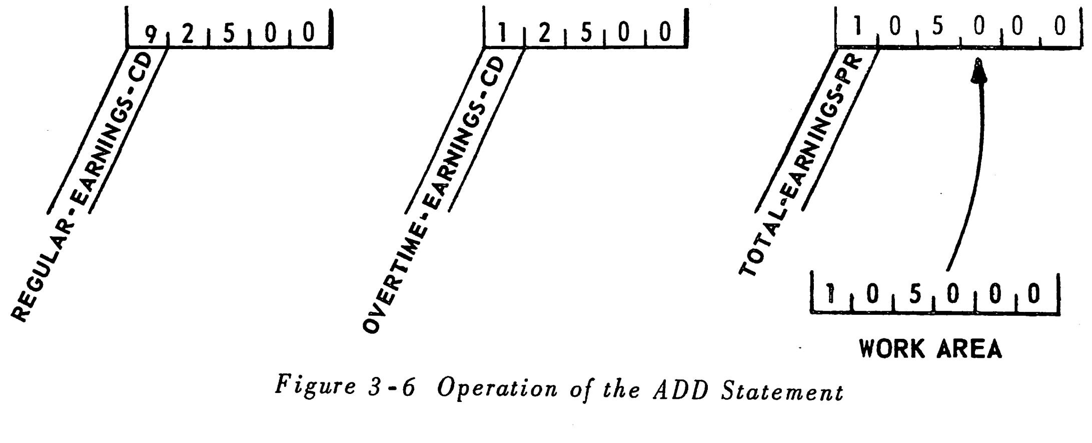

9 2 5 0 0
REGULAR-EARNINGS-CD
1 2 5 0 0
OVERTIME-EARNINGS-CD
1 0 5 0 0 0
TOTAL-EARNINGS-PR
1 0 5 0 0 0
WORK AREA

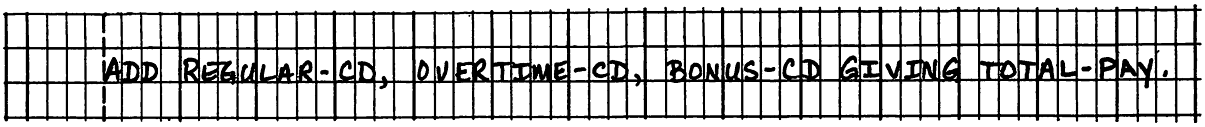
Figure 3-6 Operation of the ADD Statement

Note that the areas referenced by REGULAR-EARNINGS-CD and OVERTIME-EARNINGS-CD are not altered.

It should again be emphasized the Step 1 and Step 2 in the example in Figure 3-6 take place with the execution of the ADD statement using the GIVING option.

The following example illustrates an operation in which three factors are added together with the answer stored in the area referenced by the identifier following the GIVING.

EXAMPLE



```
ADD REGULAR-CD, OVERTIME-CD, BONUS-CD GIVING TOTAL-PAY.
```

Figure 3-7 Example of ADD Statement Using Three Factors

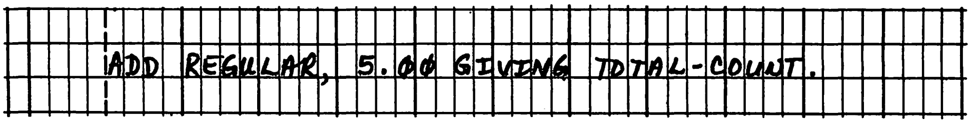
In the ADD statement in Figure 3-7 REGULAR-CD, OVERTIME-CD, and BONUS-CD are added together and the answer is stored in the area referenced by TOTAL-PAY.

The maximum number of operands that may be specified is 23. The maximum size of any amount used in an arithmetic operation is 18 decimal digits.

ADD Statement – Numeric Literals

As can be seen from the COBOL format notation for the Add statement, a "numeric literal" may be used as a part of the ADD statement. A numeric literal is nothing more than a "constant" or a number that may be used in an arithmetic statement rather than an identifier. The following example illustrates the use of a numeric literal.

EXAMPLE



```
ADD REGULAR, 5.00 GIVING TOTAL-COUNT.
```

Figure 3-8 Example of the ADD Statement with a Numeric Literal

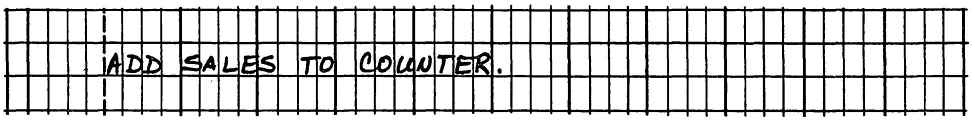
In the example the contents of the area referenced by REGULAR and 5.00 are added together and the answer is stored in the area referenced by TOTAL-COUNT.

A Numeric Literal must contain at least one and not more than 18 digits, and may consist of the characters 0 through 9, the plus or the minus sign, and the decimal point. If a Numeric Literal is unsigned, it is assumed positive. A decimal point may appear anywhere within the literal except as the rightmost character. If a numeric literal does not contain a decimal point, it is assumed to be a whole number.

ADD Statement – "TO Option"

The ADD statement may also be written using the TO option. The following example illustrates the use of this option.

EXAMPLE



```
ADD SALES TO COUNTER.
```

Figure 3-9 Example of the ADD Statement using the "TO" Option

The format notation of the ADD statement using the "TO" option is illustrated below.

$$\underline{\text{ADD}} \left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\} \left[\begin{array}{l} \text{identifier-2} \\ \text{literal-2} \end{array} \right] \ldots \; \underline{\text{TO}} \text{ identifier-m } [\underline{\text{ROUNDED}}]$$

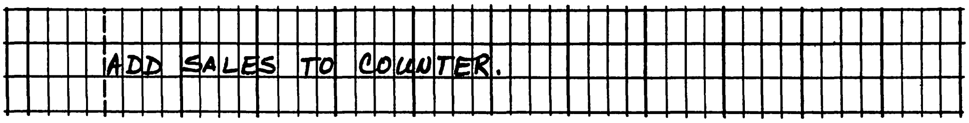
$$[\text{identifier-n } [\underline{\text{ROUNDED}}]] \ldots \; [\text{ON } \underline{\text{SIZE}} \; \underline{\text{ERROR}} \text{ imperative-statement}]$$

Figure 3-10 ADD Statement with "TO" Option – Format Notation

When the TO option is used in an arithmetic statement, the value of all identifiers (including identifier-m) and literals are added, and the resulting sum replaces the value of identifier-m.

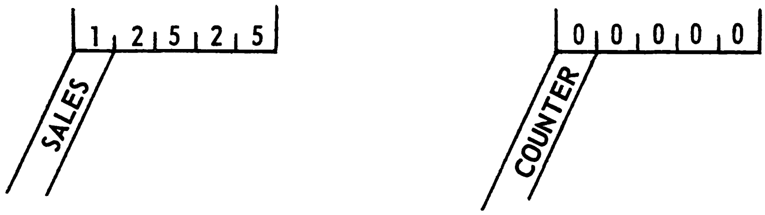
When the ADD statement in Figure 3-9 is executed, the area referenced by SALES is added to the area referenced by COUNTER and the answer is stored in the area referenced by COUNTER. The following diagrams illustrate the contents of storage before and after execution of the ADD statement.

EXAMPLE 1:



```
ADD SALES TO COUNTER.
```

Before Execution:


1 2 5 2 5
SALES
0 0 0 0 0
COUNTER

After Execution:

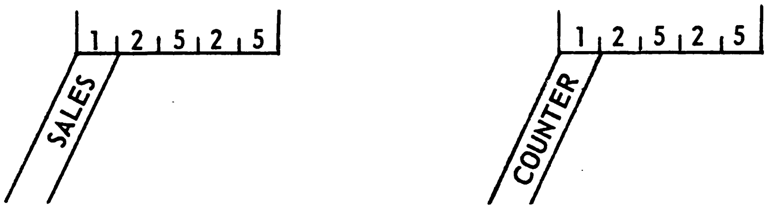
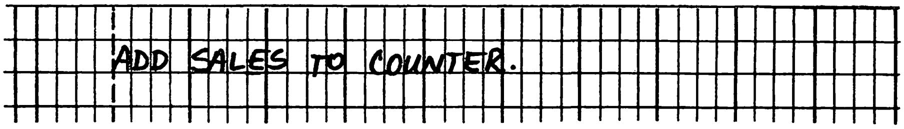

1 2 5 2 5
SALES
1 2 5 2 5
COUNTER

Figure 3-11 Example 1 of ADD Statement

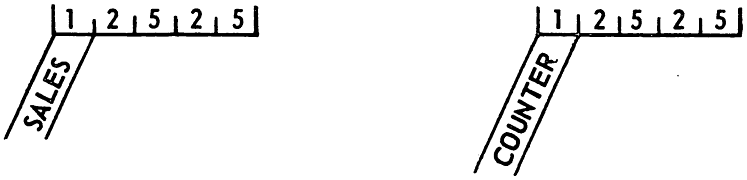
Note, that SALES is added to COUNTER and the answer is stored in COUNTER.

EXAMPLE 2:



```
ADD SALES TO COUNTER.
```

Before Execution:



After Execution:

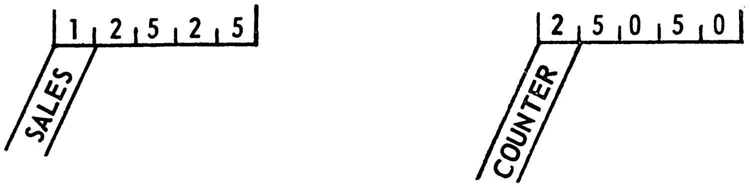
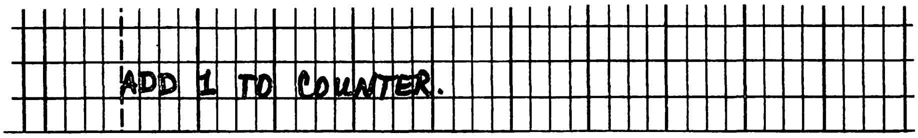


Figure 3-12 Example Operation of the ADD Statement using the "TO" Option

Note in Example 2 that SALES (which contains 12525) is added to COUNTER (which contains 12525) and the resulting answer (25050) is stored in COUNTER.

A numeric literal may also be used in the ADD statement when using the TO option. The following example illustrates the use of a numeric literal in the ADD statement using the TO option.

EXAMPLE



```
ADD 1 TO COUNTER.
```

Figure 3-13 Example of the ADD Statement using a Numeric Literal

In this example the number 1 is added to COUNTER and the answer is stored in COUNTER.

The ADD statement with the TO option is frequently used where totals are to be accumulated such as in applications requiring the preparation of reports requiring final totals.

With the ADD verb, fields are aligned according to the decimal point within each field, and all quantities are added algebracially.

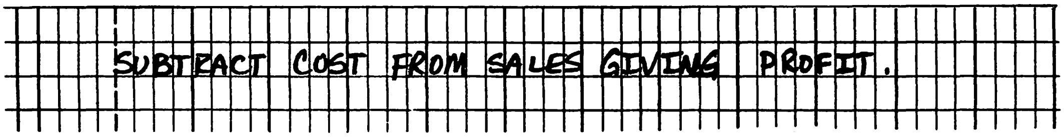
If the size of the area which is to contain the sum after calculations are performed is not large enough to contain all of the positions of the answer, significant digits of the sum will be lost and an incorrect answer will develop.

SUBTRACT STATEMENT

The SUBTRACT statement subtracts one, or a sum of two or more numeric data items from a specified item and sets the value of a data item equal to the difference.

The following example illustrates the use of the SUBTRACT statement.

EXAMPLE



```
SUBTRACT COST FROM SALES GIVING PROFIT.
```

Figure 3-14 Sample SUBTRACT Statement

In the example above, the data in the area referenced by COST is subtracted from the data in the area referenced by SALES and the answer is stored in the area referenced by PROFIT.

SUBTRACT STATEMENT – Format Notation

The basic format of the SUBTRACT is given below:

Format 1

```
              ( identifier-1 )  [identifier-2]
SUBTRACT      {              }  [            ]   ...
              ( literal-1    )  [literal-2   ]

   FROM identifier-m    [ROUNDED]

   [identifier-n [ROUNDED]] ... [ON SIZE ERROR imperative-statement]
```

Format 2

```
              ( identifier-1 )  [identifier-2 ]
SUBTRACT      {              }  [             ]   ...
              ( literal-1    )  [literal-2    ]

              ( identifier-m )
   FROM       {              }  GIVING identifier-n
              ( literal-m    )

   [ROUNDED]    [ON SIZE ERROR imperative-statement]
```

Figure 3-15 SUBTRACT Statement – Format Notation

As with the ADD statement, when subtracting using the GIVING option, the calculations are performed in a work area and the result is moved to the area referenced by the identifier following the GIVING.

When two or more operands are specified before the FROM, the effect is to add the values of all the operands that precede the FROM and then subtract this sum from the value of the item following the FROM.

EXAMPLE

```
SUBTRACT FICA, CONTRIBUTIONS FROM GROSS-PAY.
```

Figure 3-16 SUBTRACT Statement using Two Operands

When the SUBTRACT statement illustrated above is executed, the amounts referenced by FICA and CONTRIBUTIONS are added together and this sum is subtracted from GROSS-PAY and the answer is stored in the area referenced by GROSS-PAY. Note that in this example that the GIVING option is not used, and that the area referenced by GROSS-PAY will contain the answer.

SUBTRACT STATEMENT – Numeric Literals.

As with the ADD statement, numeric literals may be used when subtracting. The following example illustrates the use of a Numeric Literal with a SUBTRACT statement.

EXAMPLE

```
SUBTRACT 5.00 FROM ACCOUNT-IN GIVING NEW-BALANCE.
```

Figure 3-17 SUBTRACT Statement using a Numeric Literal

The effect of this instruction is to subtract 5.00 from ACCOUNT-IN with the answer being stored in the area referenced by NEW-BALANCE.

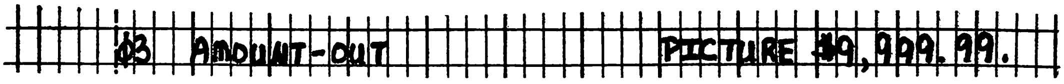
The use of the ROUNDED and ON SIZE ERROR options are explained in Chapter 4.

PICTURE CLAUSE – Basic Editing

Business reports normally require some form of editing. Editing is easily accomplished in COBOL by the use of special editing characters. To edit a field with a dollar sign, comma, and a decimal point, the dollar sign, comma, and decimal point are inserted in the Picture clause as they are to appear when the field is printed.

The following example illustrates the use of editing characters in a Picture clause.

EXAMPLE



```
03 AMOUNT-OUT                    PICTURE $9,999.99.
```

Figure 3-18 PICTURE Clause with Editing Symbols

When the program is executed, the data from the source field when moved to AMOUNT-OUT is edited according to the format established in the PICTURE clause.

The following chart illustrates data as it appears in storage, a related Picture clause, and the printed output that would result.

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
1325‸50	PICTURE $9,999.99	$1,325.50

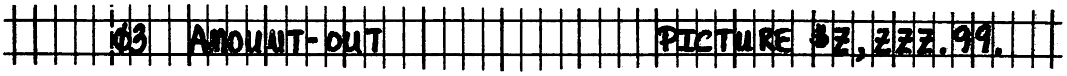
*The " ‸ " indicates the location of the assumed decimal point

Figure 3-19 PICTURE Clause and Related Printed Output

Zero Suppression

To "zero suppress" non-significant zeros, "Z's" are entered in the Picture clause up to and including the point at which zero suppression is to stop. The following example illustrates the entries in the Picture clause to zero suppress up to the decimal point.

EXAMPLE



```
03 AMOUNT-OUT                    PICTURE $Z,ZZZ.99.
```

Figure 3-20 PICTURE Clause with Zero Suppression

With the Picture clause specified above, a 6 digit number such as 032595 would print as $ 325.95 when moved to AMOUNT-OUT and printed.

PICTURE CLAUSE – Advanced Editing

There are numerous uses and variations of the PICTURE clause. The following examples present a detailed explanation of the editing capabilities of COBOL.

Zero Suppression – Asterisk Fill

Both the Z and the * represent numeric characters that are to be suppressed or replaced, when zero. The only characters that can precede either the Z or the * in a PICTURE clause are dollar sign, minus sign, plus sign, or comma.

A "Z" specifies that leading zeros of the indicated positions are to be replaced with spaces. The following examples illustrate the use of the "Z" in a PICTURE clause.

EXAMPLES: Zero Suppression

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
12595	PICTURE ZZZ99	12595
00123	PICTURE ZZZ99	123
00005	PICTURE ZZZ99	05
00000	PICTURE ZZZZZ	(blank)

Figure 3-21 Examples of Zero Suppression

This "*" (asterisk) specifies that leading zeros of the indicated positions are to be replaced by asterisks.

EXAMPLES: Asterisk Insertion

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
12345	PICTURE ***99	12345
00123	PICTURE ***99	**123
00100	PICTURE ***99	**100
00000	PICTURE ***99	***00

Figure 3-22 Examples of Asterisk Insertion

Insertion Characters

The use of an Insertion Character will cause that character to be placed in the corresponding position of the identifier specified when data is placed in that field.

When Insertion Characters are used, the PICTURE must be large enough to hold both the data and the Insertion Characters.

Of the Insertion Characters, the dollar sign, comma, and decimal point can be used only when the data in the field is numeric.

The appearance of a single symbol "$" in a PICTURE clause indicates that a dollar sign is to be inserted by the object program at the position occupied by the symbol. Several consecutive repetitions of the symbol "$" indicates that the object program is to edit the data with a floating dollar sign, which floats across the positions occupied by the consecutive symbols.

When the comma is used in the PICTURE clause, a comma is inserted by the object program in the area referenced by the data-name specified. If the zero to the left of the comma has been suppressed, then the comma will not be inserted at this position, but the character which is replacing the zeros will be inserted.

When the decimal appears in a PICTURE clause, the object program will insert a period, to serve as an actual decimal point, at the position occupied by the symbol. Data placed into the identifier specified by the object program will be aligned so that the assumed decimal point corresponds to this actual decimal point. The decimal can appear in the PICTURE clause only once. The following examples illustrate the use of the dollar sign, comma, and decimal point in PICTURE clause.

EXAMPLES: Character Insertion, Zero Suppression

DATA IN STORAGE	PICTURE	PRINTED CUTPUT
0001‸25	PICTURE $Z,ZZZ.99	$ 1.25
0000‸00	PICTURE $Z,ZZZ.99	$ .00
0003‸25	PICTURE $$,$$$.99	$3.25
0000‸00	PICTURE $$,$$$.99	$.00
0005‸50	PICTURE $*,***.99	$****5.50
0000‸05	PICTURE $*,***.99	$*****.05
‸1250	PICTURE $$,$$9.99	$0.12
1234‸	PICTURE $Z,ZZZ.99	$1,234.00

Figure 3-23 Examples of Insertion of Dollar Sign, Comma, and Decimal Point

Plus and Minus Signs

The + (plus) and - (minus) signs can be used at either end of the PICTURE. Data moved from any field that does not have a processing sign is considered positive. When the minus sign appears at either end of a PICTURE clause, the object program indicates the sign of the data in that character position. If the data is positive, a space is inserted; if the data is negative, a minus sign is inserted.

EXAMPLES: Minus Sign

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
4500	PICTURE -9999	4500
-4500	PICTURE -9999	-4500
4500	PICTURE 9999-	4500
-4500	PICTURE 9999-	4500-

Figure 3-24 Examples of Minus Signs

Report signs can only be used with numeric fields.

The minus sign can also be floated from the left-hand end of the PICTURE by placing this symbol in each leading numeric position to be suppressed. Each leading zero is replaced with a space, and either a space or a minus sign is placed in the position of the last suppressed zero.

EXAMPLES: Floating Minus Sign

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
12345	PICTURE ----99	12345
00034	PICTURE ----99	34
00000	PICTURE ----99	00
-12345	PICTURE ----99	-12345
-00045	PICTURE ----99	-45

Figure 3-25 Examples of Floating Minus Sign

When the plus sign appears at either end of a PICTURE clause, the object program indicates the sign of the data in that character position. If the data is positive, a "+" is inserted; if the data is negative, a "-" is inserted.

EXAMPLES: Plus Sign

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
12	PICTURE + 99	+12
–12	PICTURE + 99	–12
–12	PICTURE 99+	12–
12	PICTURE 99+	12+

Figure 3-26 Example of Plus Sign

The plus sign can be floated in the same way as the minus sign.

Blank

When the "B" appears in the PICTURE clause, the object program will insert a blank in each position occupied by this symbol.

EXAMPLES: B (Blank)

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
00000	PICTURE 99B999	00 000
060969	PICTURE 99B99B99	06 09 69

Figure 3-27 Example of BLANK

Credit and Debit Symbols

When the CR symbol appears at the right-hand end of a PICTURE clause, the object program indicates the sign of the data in those character positions. If the data is positive, two blanks are inserted; if the data is negative, the letters CR are inserted.

EXAMPLES: CR (Credit)

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
123‸45	PICTURE $***.99CR	$123.45
-123‸45	PICTURE $***.99CR	$123.45CR
-0123‸45	PICTURE $$,$$$.99BCR	$123.45 CR
-0045‸6	PICTURE $$,$$$.99CR	$45.60CR

Figure 3-28 Examples of CR Symbol

When the DB symbol appears at the right-hand end of a PICTURE clause, the object program indicates the sign of the data in those character positions. If the data is negative, the letters DB are inserted; if the data is positive, two blanks are inserted.

EXAMPLES: DB (Debit)

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
0012‸56	PICTURE $*,***.99DB	$***12.56
-0012‸56	PICTURE $*,***.99DB	$***12.56DB
-0012‸56	PICTURE $*,***.99BDB	$***12.56 DB

Figure 3-29 Examples of DB Symbol

The CR and DB symbols can only be used on the right end of the Picture.

Zero Symbol

When the "0" (zero) appears in the PICTURE clause, the object program will insert a zero as an additional character at the place occupied by the symbol.

EXAMPLES: 0 (Zero)

DATA IN STORAGE	PICTURE	PRINTED OUTPUT
125	PICTURE 99900	12500
125	PICTURE 00999	00125

Figure 3-30 Example of ZERO Symbol

Symbol "P"

If the assumed decimal point location is not within the field, the symbol P is used to indicate each scaling position beyond either end of identifier-1. In effect, each P represents an implied position.

EXAMPLES: P (Implied Position)

DATA IN STORAGE	PICTURE	REPRESENTATION
123	PICTURE 999PPPV	123000.
123	PICTURE VPP999	.00123

Figure 3-31 Examples of "P" Symbol

MOVE STATEMENT

The previous chapter explained the basic operation of the MOVE statement. This statement is used to move data from one area of storage to another.

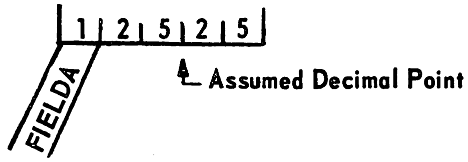
When the field from which the data is to be moved and the field or fields to receive the data are numeric by definition (Picture with 9's or editing characters), then the data is aligned by the actual or assumed decimal. Truncation or extension will take place on one or both ends of the data when it is moved to the receiving field.

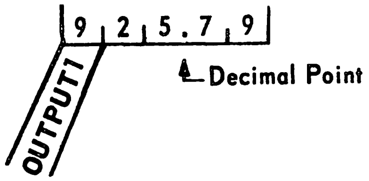
When data is moved from one area to another, the field is edited according to the editing characters specified in the receiving field. The following examples illustrate the operation of the Move statement.

EXAMPLE 1:

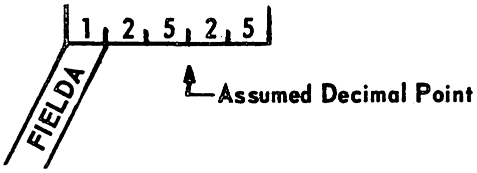
```
    03  FIELDA                      PICTURE 999V99.
    03  OUTPUT1                     PICTURE 999.99.
    MOVE FIELDA TO OUTPUT1.
```

Before Execution:


1 2 5 2 5
FIELDA
Assumed Decimal Point


9 2 5 . 7 9
OUTPUT1
Decimal Point

After Execution:


1 2 5 2 5
FIELDA
Assumed Decimal Point

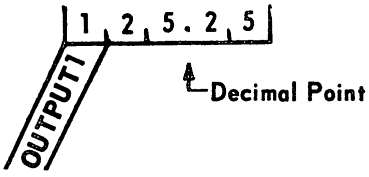

1 2 5 . 2 5
OUTPUT1
Decimal Point

Figure 3-32 Example of EDITING

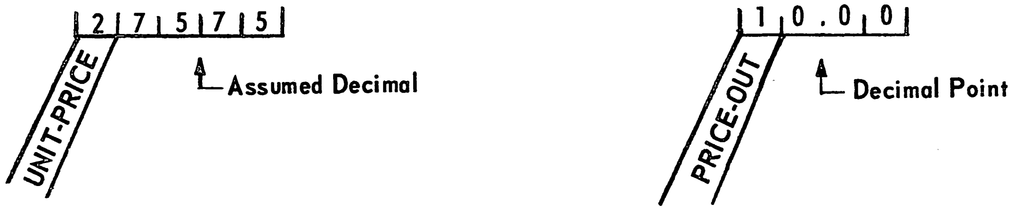
In this example, FIELDA is moved to OUTPUT1 and the data referenced in FIELDA is aligned in OUTPUT1 on the basis of the assumed and actual decimal points specified in the PICTURE clauses.

EXAMPLE 2:

In this example because only 2 positions are specified to the left of the decimal point in PRICE-OUT, the high-order digit is lost when UNIT-PRICE is moved to PRICE-OUT.

```
    03  UNIT-PRICE                PICTURE 999V99.
    03  PRICE-OUT                 PICTURE 99.99.
    MOVE UNIT-PRICE TO PRICE-OUT.
```

Before Execution:


2 7 5 7 5
UNIT-PRICE
Assumed Decimal
1 0 . 0 0
PRICE-OUT
Decimal Point

After Execution:

Note that the left-most position is truncated after execution of the Move with decimal alignment.

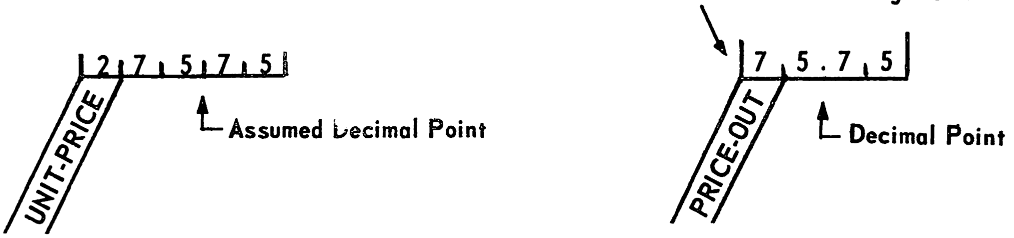

2 7 5 7 5
UNIT-PRICE
Assumed Decimal Point
7 5 . 7 5
PRICE-OUT
Decimal Point

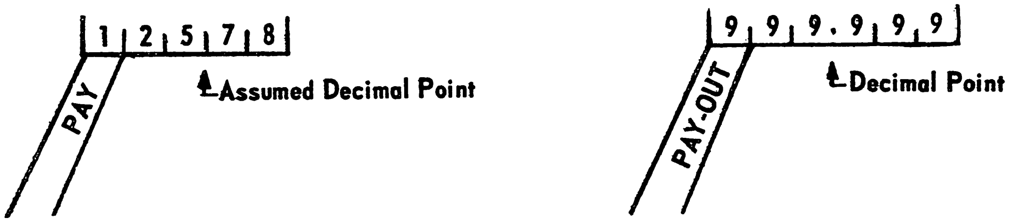
Figure 3-33 Example of Truncation when Editing

EXAMPLE 3:

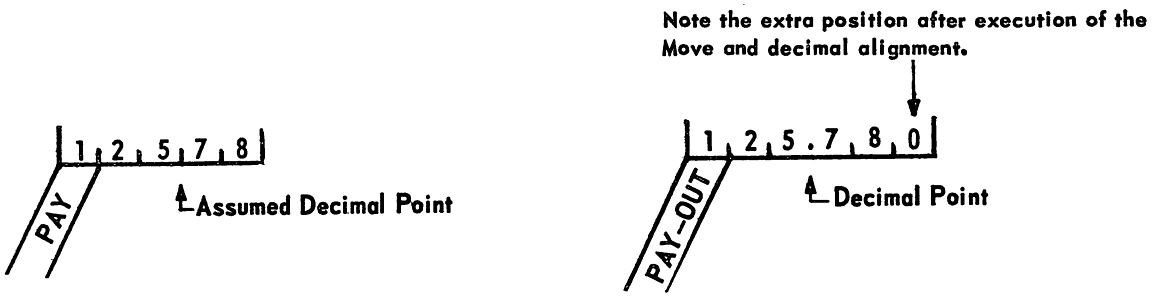
In the example below the sending field, PAY, has two positions to the right of the decimal point, and the receiving field, PAY-OUT, has three positions to the right of the decimal point as specified in the PICTURE clauses. When the MOVE statement is executed, the receiving field PAY-OUT will have the third position to the right of the decimal point extended with a zero.

```
    03  PAY                      PICTURE 999V99.
    03  PAY-OUT                  PICTURE 999.999.
    MOVE PAY TO PAY-OUT.
```

Before Execution:


1 2 5 7 8
PAY
Assumed Decimal Point
9 9 9 . 9 9 9
PAY-OUT
Decimal Point

After Execution:


Note the extra position after execution of the Move and decimal alignment.

1 2 5 7 8
PAY
Assumed Decimal Point
1 2 5 . 7 8 0
PAY-OUT
Decimal Point

Figure 3-34 Example of Different Length Fields

Normally the fields will be of equal lengths; however, the programmer should have an understanding of the internal operation of instructions.

SAMPLE PROBLEM

The following pages illustrate the complete documentation for the sample problem developed in this chapter. Input consists of Employee Earnings cards, containing the Employee Number, Employee Name, Regular Earnings, and Overtime Earnings. Total Earnings are to be calculated by adding Regular Earnings to Overtime Earnings. If there are no Overtime Earnings, the field will contain zeros.

Output is to consist of a printed report.

INPUT

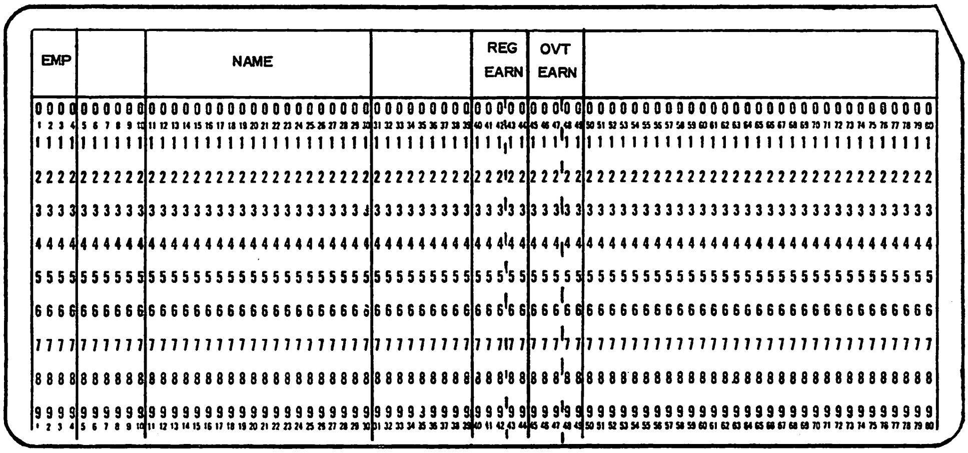
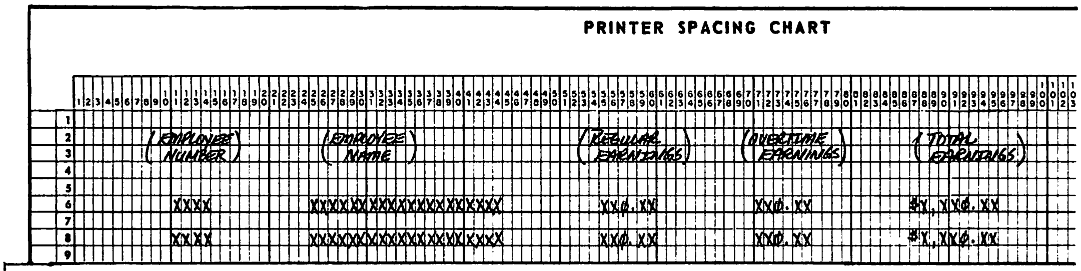


Figure 3-35 Input Card

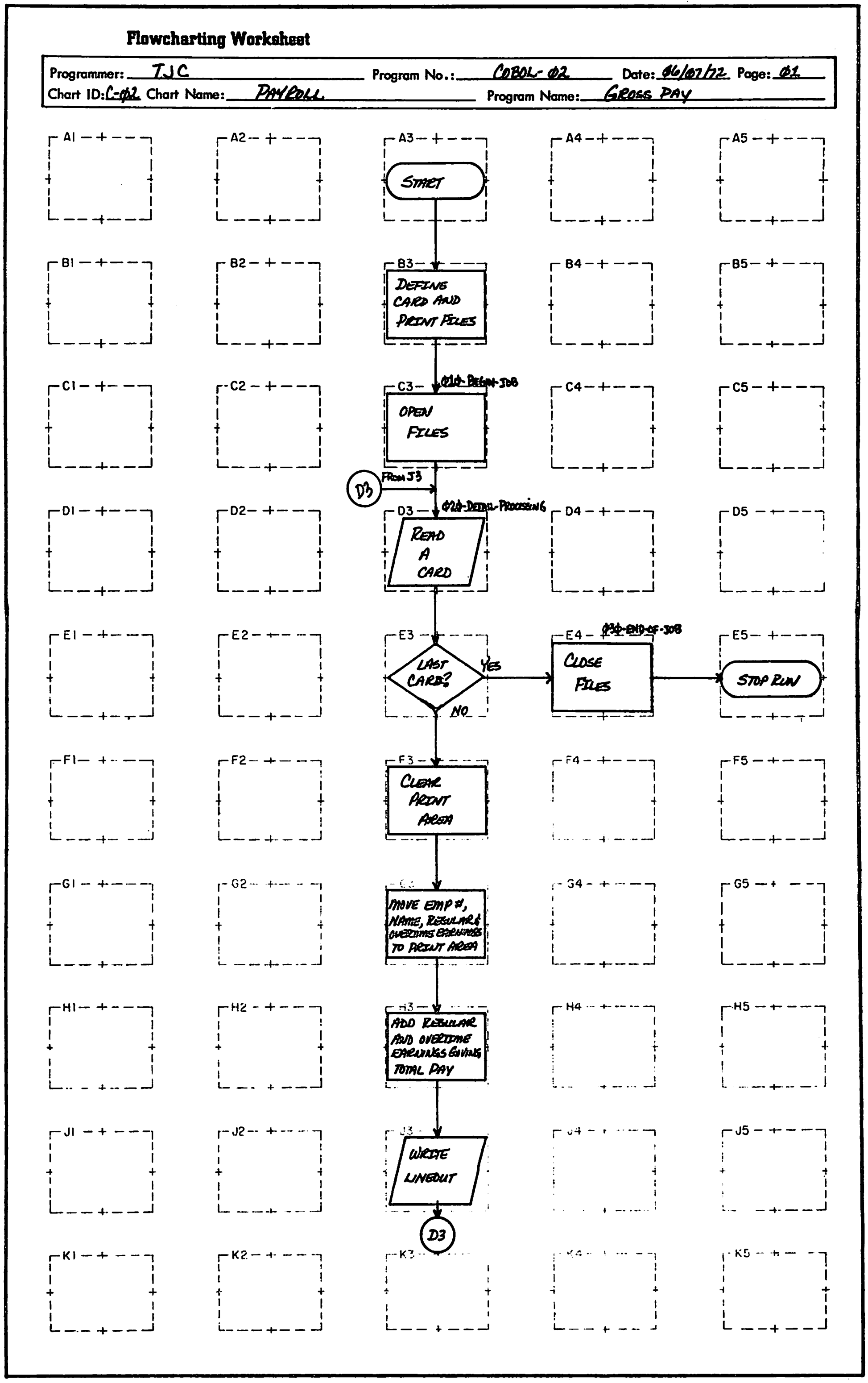
OUTPUT


PRINTER SPACING CHART
(EMPLOYEE NUMBER)
(EMPLOYEE NAME)
(REGULAR EARNINGS)
(OVERTIME EARNINGS)
(TOTAL EARNINGS)
XXXX
XXXXXXXXXXXXXXXXXXXX
XXØ.XX
XXØ.XX
$X,XXØ.XX
XXXX
XXXXXXXXXXXXXXXXXXXX
XXØ.XX
XXØ.XX
$X,XXØ.XX

```
0990     AARON, WILLIAM        900.00     100.00    $1,000.00
1004     DECKER, WILLIAM        90.00       5.00    $   95.00
1005     EVANS, MARY           100.00        .00    $  100.C0
1006     HUGH, GUY             125.25      50.00    $  175.25
1008     KICK,JIM              100.25      40.00    $  140.25
1007     JAY, JANE             175.25      55.00    $  230.25
1009     LANG,JAMES            105.50     100.25    $  205.75
1010     MOORE,TOM             124.25      60.00    $  184.25
1011     NORSE,JERRY           125.25      50.00    $  175.25
1012     OVER, TIM             105.50     100.00    $  205.50
1013     REEL,KELLY            125.50      50.55    $  176.05
1014     ROBERTS,JIM           125.25      50.00    $  175.25
```

Figure 3-36 Printed Output

FLOWCHART



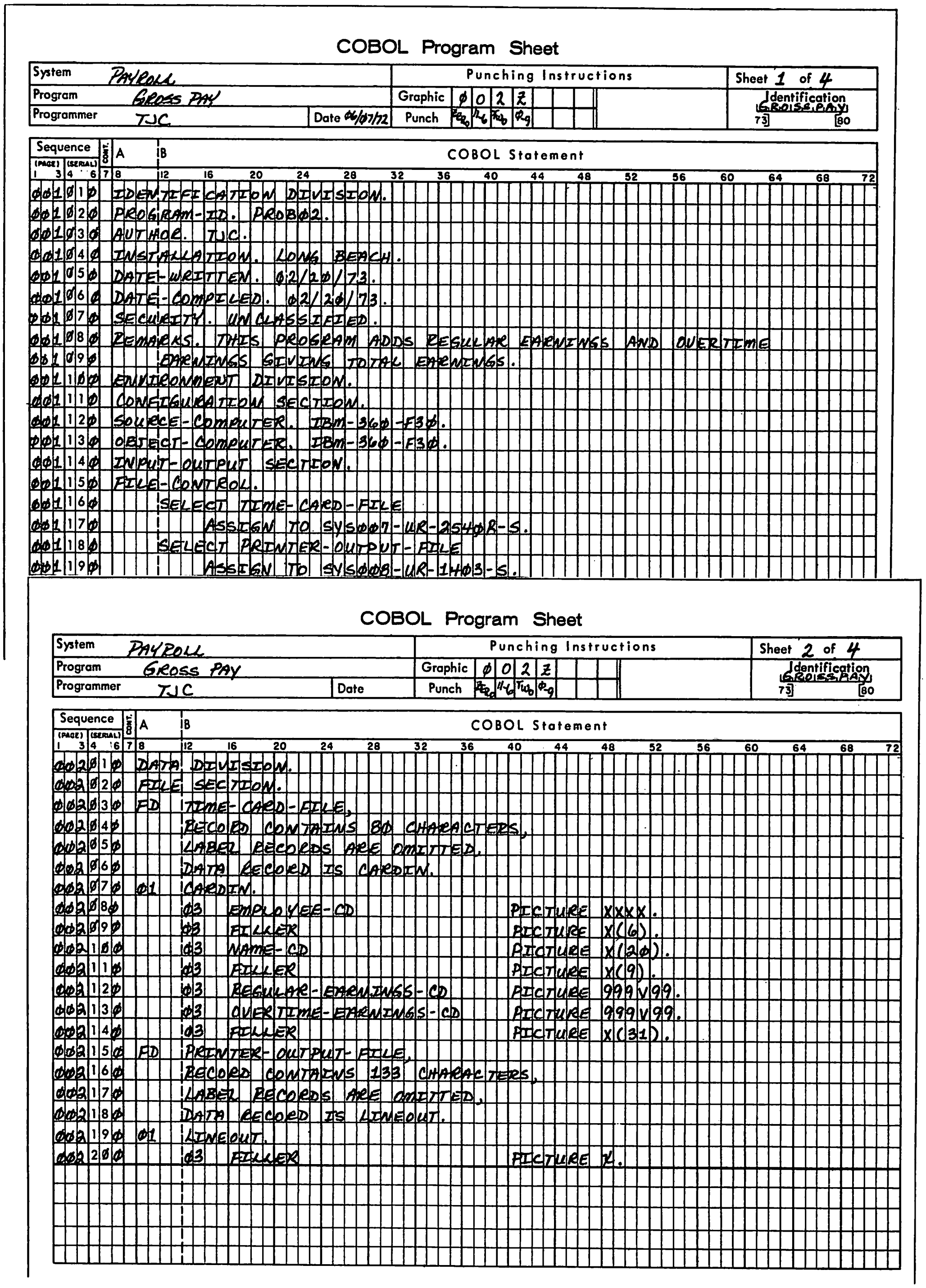
Flowcharting Worksheet

Programmer: TJC Program No.: COBOL-02 Date: 06/07/72 Page: 01
Chart ID: C-02 Chart Name: PAYROLL Program Name: GROSS PAY

START
DEFINE CARD AND PRINT FILES
010-BEGIN-JOB
OPEN FILES
D3
FROM J3
020-DETAIL-PROCESSING
READ A CARD
LAST CARD?
YES
NO
030-END-OF-JOB
CLOSE FILES
STOP RUN
CLEAR PRINT AREA
MOVE EMP #, NAME, REGULAR & OVERTIME EARNINGS TO PRINT AREA
ADD REGULAR AND OVERTIME EARNINGS GIVING TOTAL PAY
WRITE LINEOUT
D3

Figure 3-37 Program Flowchart

Source Coding – Sample Program



COBOL Program Sheet

System PAYROLL
Program GROSS PAY
Programmer TJC
Date 06/07/72
Punching Instructions
Graphic 0 O 2 Z
Punch Zero 11-6 Two 0-9
Sheet 1 of 4
Identification GROSSPAY 73 80

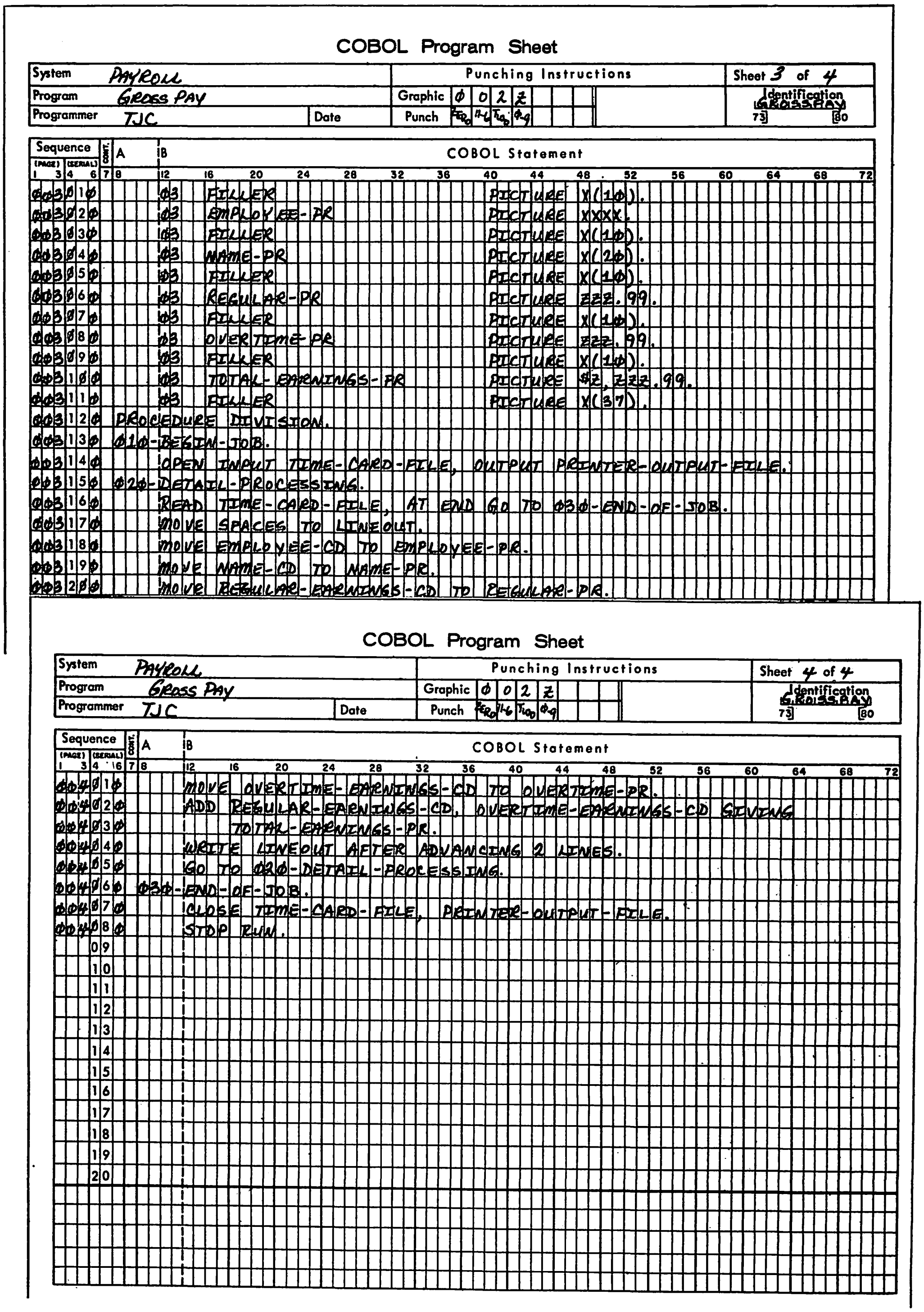
```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. PROB02.
001030 AUTHOR. TJC.
001040 INSTALLATION. LONG BEACH.
001050 DATE-WRITTEN. 02/20/73.
001060 DATE-COMPILED. 02/20/73.
001070 SECURITY. UNCLASSIFIED.
001080 REMARKS. THIS PROGRAM ADDS REGULAR EARNINGS AND OVERTIME
001090     EARNINGS GIVING TOTAL EARNINGS.
001100 ENVIRONMENT DIVISION.
001110 CONFIGURATION SECTION.
001120 SOURCE-COMPUTER. IBM-360-F30.
001130 OBJECT-COMPUTER. IBM-360-F30.
001140 INPUT-OUTPUT SECTION.
001150 FILE-CONTROL.
001160     SELECT TIME-CARD-FILE
001170             ASSIGN TO SYS007-UR-2540R-S.
001180     SELECT PRINTER-OUTPUT-FILE
001190             ASSIGN TO SYS008-UR-1403-S.
```

COBOL Program Sheet

System PAYROLL
Program GROSS PAY
Programmer TJC
Date
Punching Instructions
Graphic 0 O 2 Z
Punch Zero 11-6 Two 0-9
Sheet 2 of 4
Identification GROSSPAY 73 80

```
002010 DATA DIVISION.
002020 FILE SECTION.
002030 FD  TIME-CARD-FILE,
002040     RECORD CONTAINS 80 CHARACTERS,
002050     LABEL RECORDS ARE OMITTED,
002060     DATA RECORD IS CARDIN.
002070 01  CARDIN.
002080     03  EMPLOYEE-CD                 PICTURE XXXX.
002090     03  FILLER                      PICTURE X(6).
002100     03  NAME-CD                     PICTURE X(20).
002110     03  FILLER                      PICTURE X(9).
002120     03  REGULAR-EARNINGS-CD         PICTURE 999V99.
002130     03  OVERTIME-EARNINGS-CD        PICTURE 999V99.
002140     03  FILLER                      PICTURE X(31).
002150 FD  PRINTER-OUTPUT-FILE,
002160     RECORD CONTAINS 133 CHARACTERS,
002170     LABEL RECORDS ARE OMITTED,
002180     DATA RECORD IS LINEOUT.
002190 01  LINEOUT.
002200     03  FILLER                      PICTURE X.
```

Figure 3-38 Sample Program – Part 1 of 2



COBOL Program Sheet

System PAYROLL
Program GROSS PAY
Programmer TJC
Date

Punching Instructions
Graphic Ø 0 2 Z
Punch Zero 11-6 Two 0-9

Sheet 3 of 4
Identification GROSSPAY
73 80

Sequence (PAGE) (SERIAL) CONT. A B COBOL Statement

```
003010     03  FILLER                   PICTURE X(10).
003020     03  EMPLOYEE-PR              PICTURE XXXX.
003030     03  FILLER                   PICTURE X(10).
003040     03  NAME-PR                  PICTURE X(20).
003050     03  FILLER                   PICTURE X(10).
003060     03  REGULAR-PR               PICTURE ZZZ.99.
003070     03  FILLER                   PICTURE X(10).
003080     03  OVERTIME-PR              PICTURE ZZZ.99.
003090     03  FILLER                   PICTURE X(10).
003100     03  TOTAL-EARNINGS-PR        PICTURE $Z,ZZZ.99.
003110     03  FILLER                   PICTURE X(37).
003120 PROCEDURE DIVISION.
003130 010-BEGIN-JOB.
003140     OPEN INPUT TIME-CARD-FILE, OUTPUT PRINTER-OUTPUT-FILE.
003150 020-DETAIL-PROCESSING.
003160     READ TIME-CARD-FILE, AT END GO TO 030-END-OF-JOB.
003170     MOVE SPACES TO LINEOUT.
003180     MOVE EMPLOYEE-CD TO EMPLOYEE-PR.
003190     MOVE NAME-CD TO NAME-PR.
003200     MOVE REGULAR-EARNINGS-CD TO REGULAR-PR.
```

COBOL Program Sheet

System PAYROLL
Program GROSS PAY
Programmer TJC
Date

Punching Instructions
Graphic Ø 0 2 Z
Punch Zero 11-6 Two 0-9

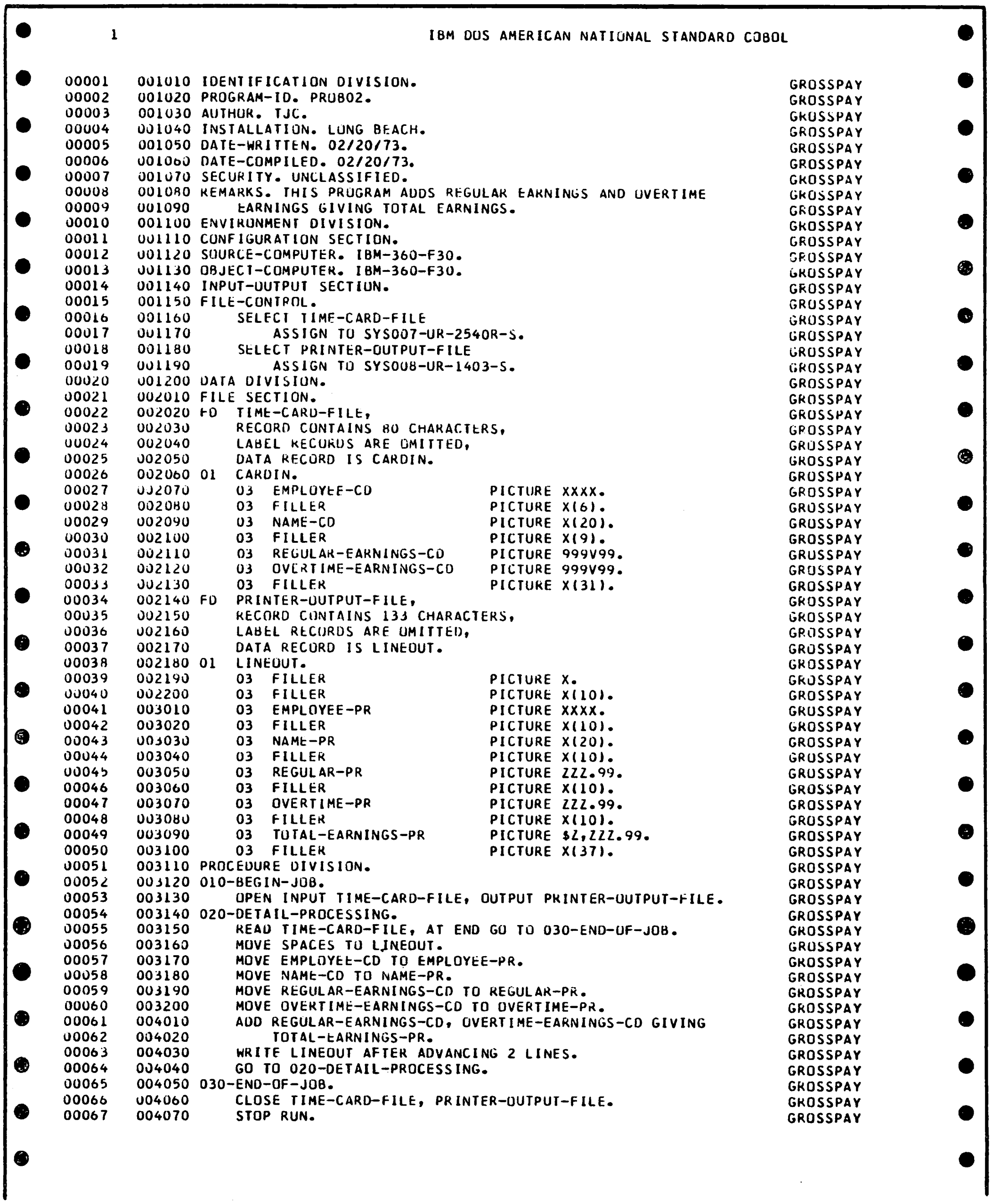
Sheet 4 of 4
Identification GROSSPAY
73 80

Sequence (PAGE) (SERIAL) CONT. A B COBOL Statement

```
004010     MOVE OVERTIME-EARNINGS-CD TO OVERTIME-PR.
004020     ADD REGULAR-EARNINGS-CD, OVERTIME-EARNINGS-CD GIVING
004030         TOTAL-EARNINGS-PR.
004040     WRITE LINEOUT AFTER ADVANCING 2 LINES.
004050     GO TO 020-DETAIL-PROCESSING.
004060 030-END-OF-JOB.
004070     CLOSE TIME-CARD-FILE, PRINTER-OUTPUT-FILE.
004080     STOP RUN.
   09
   10
   11
   12
   13
   14
   15
   16
   17
   18
   19
   20
```

Figure 3-39 Sample Program – Part 2 of 2

Source Listing – Sample Program



```
    1                                   IBM DOS AMERICAN NATIONAL STANDARD COBOL

00001   001010 IDENTIFICATION DIVISION.                                         GROSSPAY
00002   001020 PROGRAM-ID. PROB02.                                              GROSSPAY
00003   001030 AUTHOR. TJC.                                                     GROSSPAY
00004   001040 INSTALLATION. LONG BEACH.                                        GROSSPAY
00005   001050 DATE-WRITTEN. 02/20/73.                                          GROSSPAY
00006   001060 DATE-COMPILED. 02/20/73.                                         GROSSPAY
00007   001070 SECURITY. UNCLASSIFIED.                                          GROSSPAY
00008   001080 REMARKS. THIS PROGRAM ADDS REGULAR EARNINGS AND OVERTIME         GROSSPAY
00009   001090     EARNINGS GIVING TOTAL EARNINGS.                              GROSSPAY
00010   001100 ENVIRONMENT DIVISION.                                            GROSSPAY
00011   001110 CONFIGURATION SECTION.                                           GROSSPAY
00012   001120 SOURCE-COMPUTER. IBM-360-F30.                                    GROSSPAY
00013   001130 OBJECT-COMPUTER. IBM-360-F30.                                    GROSSPAY
00014   001140 INPUT-OUTPUT SECTION.                                            GROSSPAY
00015   001150 FILE-CONTROL.                                                    GROSSPAY
00016   001160     SELECT TIME-CARD-FILE                                        GROSSPAY
00017   001170         ASSIGN TO SYS007-UR-2540R-S.                             GROSSPAY
00018   001180     SELECT PRINTER-OUTPUT-FILE                                   GROSSPAY
00019   001190         ASSIGN TO SYS008-UR-1403-S.                              GROSSPAY
00020   001200 DATA DIVISION.                                                   GROSSPAY
00021   002010 FILE SECTION.                                                    GROSSPAY
00022   002020 FD  TIME-CARD-FILE,                                              GROSSPAY
00023   002030     RECORD CONTAINS 80 CHARACTERS,                               GROSSPAY
00024   002040     LABEL RECORDS ARE OMITTED,                                   GROSSPAY
00025   002050     DATA RECORD IS CARDIN.                                       GROSSPAY
00026   002060 01  CARDIN.                                                      GROSSPAY
00027   002070     03  EMPLOYEE-CD             PICTURE XXXX.                    GROSSPAY
00028   002080     03  FILLER                  PICTURE X(6).                    GROSSPAY
00029   002090     03  NAME-CD                 PICTURE X(20).                   GROSSPAY
00030   002100     03  FILLER                  PICTURE X(9).                    GROSSPAY
00031   002110     03  REGULAR-EARNINGS-CD     PICTURE 999V99.                  GROSSPAY
00032   002120     03  OVERTIME-EARNINGS-CD    PICTURE 999V99.                  GROSSPAY
00033   002130     03  FILLER                  PICTURE X(31).                   GROSSPAY
00034   002140 FD  PRINTER-OUTPUT-FILE,                                         GROSSPAY
00035   002150     RECORD CONTAINS 133 CHARACTERS,                              GROSSPAY
00036   002160     LABEL RECORDS ARE OMITTED,                                   GROSSPAY
00037   002170     DATA RECORD IS LINEOUT.                                      GROSSPAY
00038   002180 01  LINEOUT.                                                     GROSSPAY
00039   002190     03  FILLER                  PICTURE X.                       GROSSPAY
00040   002200     03  FILLER                  PICTURE X(10).                   GROSSPAY
00041   003010     03  EMPLOYEE-PR             PICTURE XXXX.                    GROSSPAY
00042   003020     03  FILLER                  PICTURE X(10).                   GROSSPAY
00043   003030     03  NAME-PR                 PICTURE X(20).                   GROSSPAY
00044   003040     03  FILLER                  PICTURE X(10).                   GROSSPAY
00045   003050     03  REGULAR-PR              PICTURE ZZZ.99.                  GROSSPAY
00046   003060     03  FILLER                  PICTURE X(10).                   GROSSPAY
00047   003070     03  OVERTIME-PR             PICTURE ZZZ.99.                  GROSSPAY
00048   003080     03  FILLER                  PICTURE X(10).                   GROSSPAY
00049   003090     03  TOTAL-EARNINGS-PR       PICTURE $Z,ZZZ.99.               GROSSPAY
00050   003100     03  FILLER                  PICTURE X(37).                   GROSSPAY
00051   003110 PROCEDURE DIVISION.                                              GROSSPAY
00052   003120 010-BEGIN-JOB.                                                   GROSSPAY
00053   003130     OPEN INPUT TIME-CARD-FILE, OUTPUT PRINTER-OUTPUT-FILE.       GROSSPAY
00054   003140 020-DETAIL-PROCESSING.                                           GROSSPAY
00055   003150     READ TIME-CARD-FILE, AT END GO TO 030-END-OF-JOB.            GROSSPAY
00056   003160     MOVE SPACES TO LINEOUT.                                      GROSSPAY
00057   003170     MOVE EMPLOYEE-CD TO EMPLOYEE-PR.                             GROSSPAY
00058   003180     MOVE NAME-CD TO NAME-PR.                                     GROSSPAY
00059   003190     MOVE REGULAR-EARNINGS-CD TO REGULAR-PR.                      GROSSPAY
00060   003200     MOVE OVERTIME-EARNINGS-CD TO OVERTIME-PR.                    GROSSPAY
00061   004010     ADD REGULAR-EARNINGS-CD, OVERTIME-EARNINGS-CD GIVING         GROSSPAY
00062   004020         TOTAL-EARNINGS-PR.                                       GROSSPAY
00063   004030     WRITE LINEOUT AFTER ADVANCING 2 LINES.                       GROSSPAY
00064   004040     GO TO 020-DETAIL-PROCESSING.                                 GROSSPAY
00065   004050 030-END-OF-JOB.                                                  GROSSPAY
00066   004060     CLOSE TIME-CARD-FILE, PRINTER-OUTPUT-FILE.                   GROSSPAY
00067   004070     STOP RUN.                                                    GROSSPAY
```

Figure 3-40 Source Listing

CHAPTER 3

PROGRAMMING ASSIGNMENT 1

ADDITION

INSTRUCTIONS:

From the Sales cards illustrated below a Current Sales Report is to be prepared. Total Sales are to be calculated by adding Y-T-D Sales and Current Sales. The Salesman Number, Salesman Name, Y-T-D Sales, Current Sales, and Total Sales are to be printed.

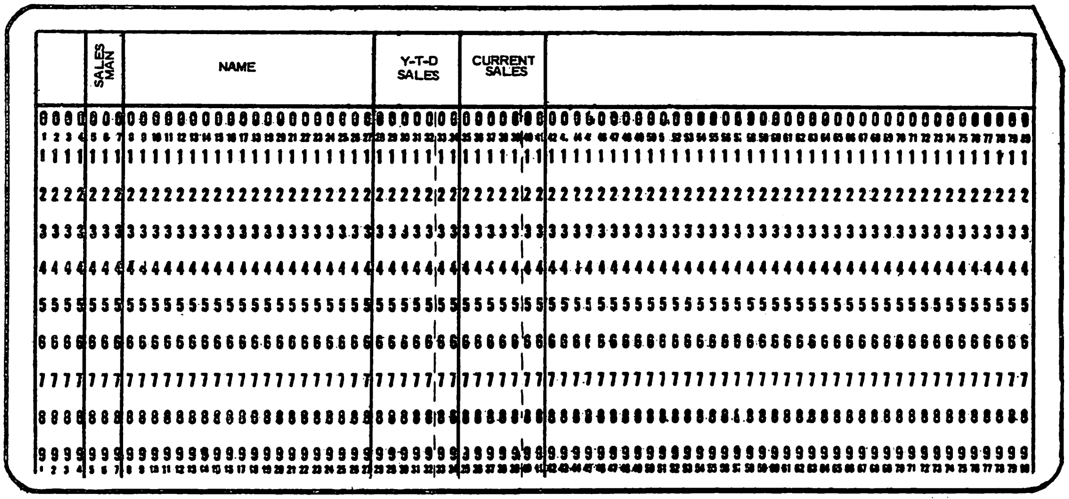
Salesman Number is to be zero suppressed up to the low-order digit. Y-T-D Sales and Current Sales are to be zero suppressed and edited with a comma and a decimal point. Total Sales is to be zero suppressed and edited with a dollar sign, comma, and a decimal point.

Cards are to be sorted by Salesman Number.

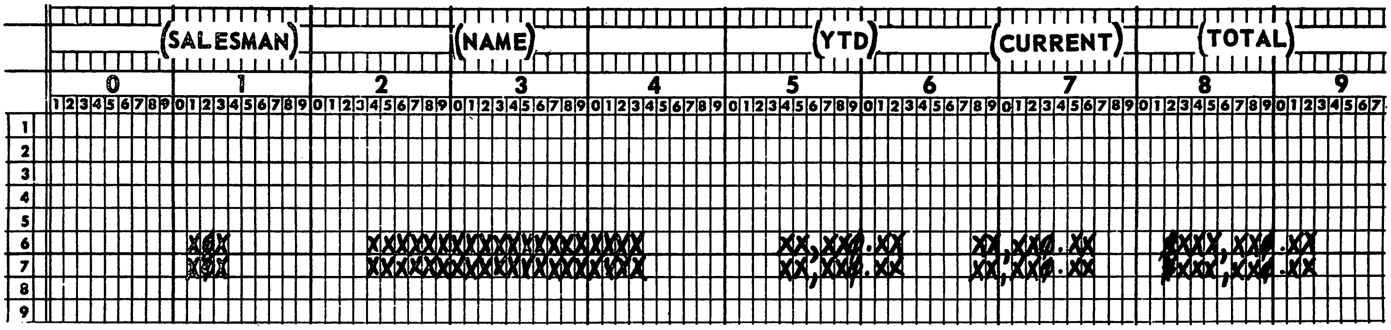
Prepare a flowchart and the COBOL program to produce the required report.

Test Data is contained in the Appendices.

INPUT: Sales Cards


SALES MAN
NAME
Y-T-D SALES
CURRENT SALES

OUTPUT: Current Sales Report


(SALESMAN)
(NAME)
(YTD)
(CURRENT)
(TOTAL)
XXX
XXXXXXXXXXXXXXXXXXXX
XX,XX0.XX
XX,XX0.XX
$XXX,XX0.XX

CHAPTER 3

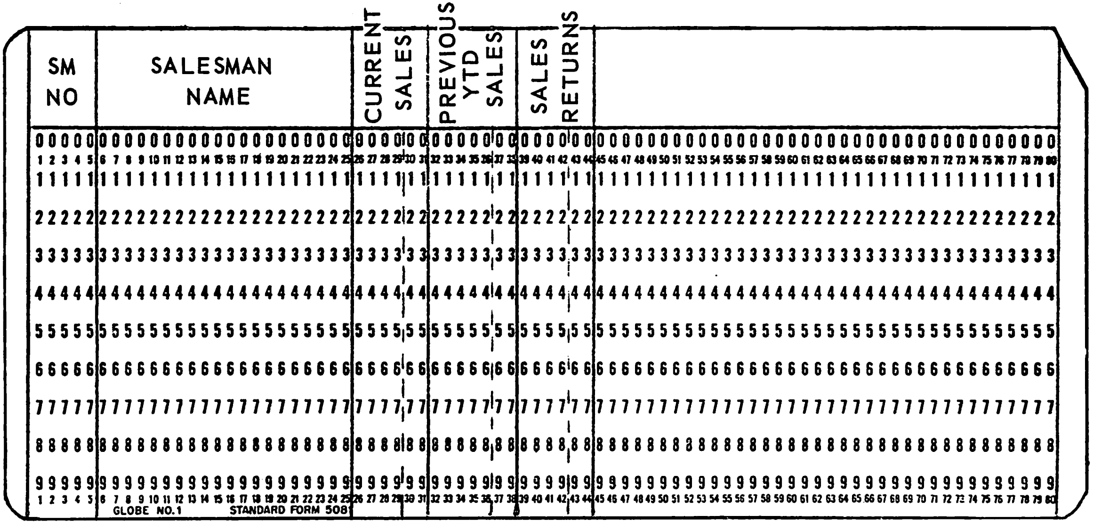
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

Write the COBOL program to produce a Sales Listing Report. Use the test data on page 3.30.

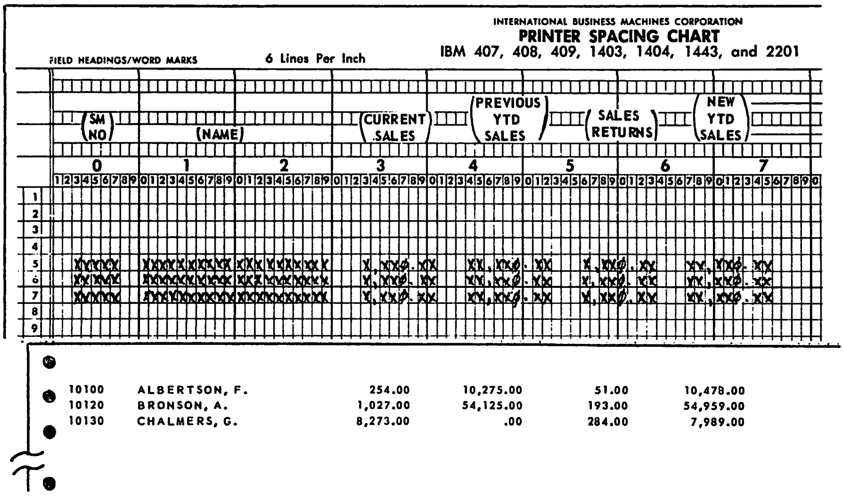
INPUT

The input consists of Sales Cards which contain the Salesman Number, the Salesman Name, the Current Sales, the Previous Year-To-Date Sales, and the Sales Returns.


SM NO
SALESMAN NAME
CURRENT SALES
PREVIOUS YTD SALES
SALES RETURNS
GLOBE NO.1
STANDARD FORM 508

OUTPUT

The output is to consist of a Sales Listing which contains the Salesman Number, the Salesman Name, the Current Sales, the Previous Year-To-Date Sales, the Sales Returns, and the New Year-To-Date Sales. The New Year-To-Date Sales are calculated by adding the Current Sales and the Previous Year-To-Date Sales and then subtracting the Sales Returns.



INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, and 2201

FIELD HEADINGS/WORD MARKS
6 Lines Per Inch

SM NO
(NAME)
CURRENT SALES
PREVIOUS YTD SALES
SALES RETURNS
NEW YTD SALES

```
 XXXXX   XXXXXXXXXXXXXXXXXXXX     X,XX0.XX     XX,XX0.XX     X,XX0.XX     XX,XX0.XX
 XXXXX   XXXXXXXXXXXXXXXXXXXX     X,XX0.XX     XX,XX0.XX     X,XX0.XX     XX,XX0.XX
 XXXXX   XXXXXXXXXXXXXXXXXXXX     X,XX0.XX     XX,XX0.XX     X,XX0.XX     XX,XX0.XX
```

```
10100    ALBERTSON, F.            254.00     10,275.00        51.00     10,478.00
10120    BRONSON, A.            1,027.00     54,125.00       193.00     54,959.00
10130    CHALMERS, G.           8,273.00           .00       284.00      7,989.00
```

TEST DATA – ASSIGNMENT NO. 2

Salesman Number (col 1-5)	Name (col 6-25)	Current Sales (col 26-31)	Previous YTD Sales (col 32-38)	Sales Returns (col 39-44)
10100	Albertson, F.	025400	1027500	005100
10120	Bronson, A.	102700	5412500	019300
10130	Chalmers, G.	827300	0000000	028400
10140	Delvertz, H.	398700	3890600	003250
10150	Fritz, C.	653098	3452876	013290
10160	Lilum, R.	126750	0387690	003189

CHAPTER 4

MULTIPLICATION – DIVISION

INTRODUCTION

In performing any multiplication operation, manually or with a computer, the maximum size of the answer that may develop may be determined by adding the number of digits in the multiplier to the number of digits in the multiplicand. For example, if multiplying a 3 digit Rate field by a 2 digit Hours field, the maximum size of the answer that may develop is an answer 5 digits in length.

EXAMPLE:

$3.20 RATE (Multiplicand)
40 HOURS (Multiplier)
$128.00 (Maximum size answer-5 digits in length)

It is important that the programmer understand this basic concept of arithmetic to properly perform multiplication operations using the computer.

Rounding

In programming for business applications involving decimal positions in the answer it is frequently desirable to round off the answer. For example, in a problem if the answer is developed as: $1.254 it may be desirable to round the answer to $1.25 so that the amount can be expressed in dollars and cents. If the answer is developed as $1.255 the amount would be rounded upward to $1.26. Note in the preceding examples that if the low order position is less than 5 the amount is not rounded upward. If the low order position is 5 or greater, the amount is rounded upward.

Rounding is normally accomplished when programming using "assembler type" programming languages for business applications by adding 5 to the right of the last position to be retained and then dropping this position.

EXAMPLE:

$125.254 ←
5
125.25$\cancel{9}$ Drop the low order position. Note that when the low order digit is less than 5 rounding does not occur.

125.256 ←
5
125.26$\cancel{1}$ Drop the low order position. Note that rounding upward has occurred as the low order digit was greater than 5.

With COBOL rounding is accomplished by the use of the "ROUNDED" option available with all arithmetic statements.

SAMPLE PROBLEM

The sample program developed in this chapter illustrates an application in which input data cards containing an Employee Number Field, an Employee Name Field, an Hours Field, and a Rate Field are used to prepare a printed report. The format of the input cards is illustrated below.

Input

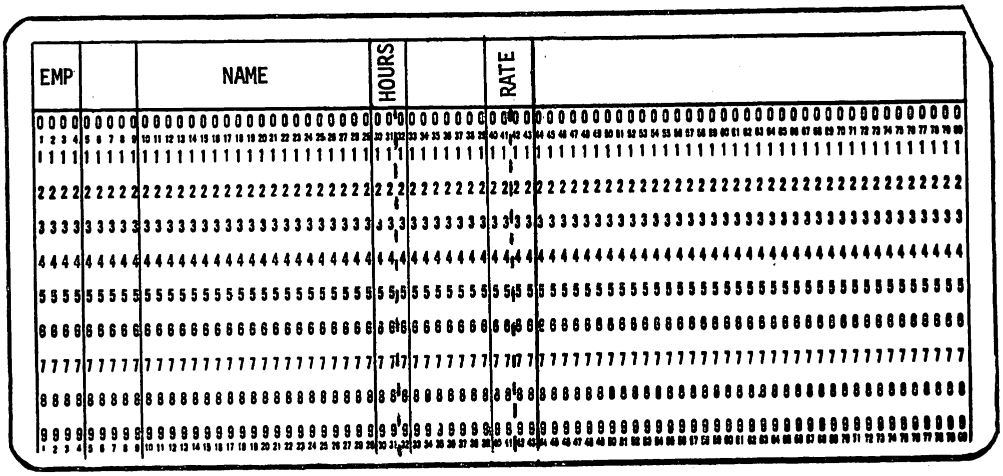
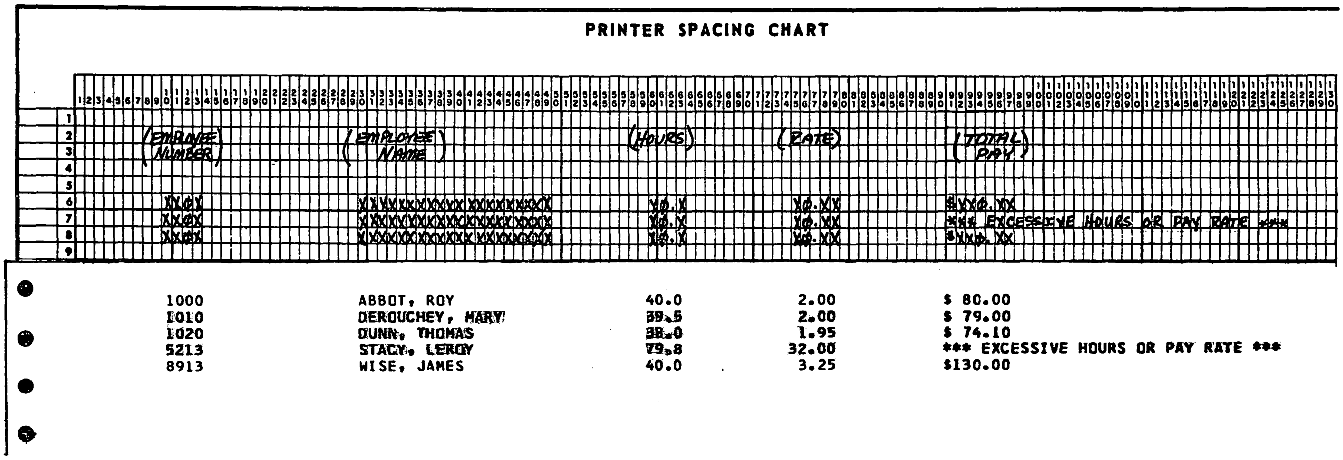

EMP
NAME
HOURS
RATE

Figure 4-1 Input Card

The output report is obtained by printing all of the fields on the card. In addition, the Net Pay is to be printed on the report. Net Pay is obtained by multiplying the Rate Field by the Hours Field and rounding off the answer.

In the sample problem Net Pay should never exceed $999.999. If Net Pay exceeds this amount, a message ***EXCESSIVE HOURS OR PAY RATE*** is to be printed on this report.

Output

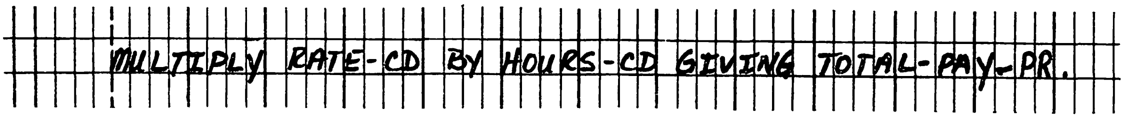

PRINTER SPACING CHART
(EMPLOYEE NUMBER)
(EMPLOYEE NAME)
(HOURS)
(RATE)
(TOTAL PAY)
XX@X
XXXXXXXXXXXXXXXXXXXX
X@.X
X@.XX
$XX@.XX
*** EXCESSIVE HOURS OR PAY RATE ***

```
        1000          ABBOT, ROY                40.0     2.00     $ 80.00
        1010          DEROUCHEY, MARY           39.5     2.00     $ 79.00
        1020          DUNN, THOMAS              38.0     1.95     $ 74.10
        5213          STACY, LEROY              79.8    32.00     *** EXCESSIVE HOURS OR PAY RATE ***
        8913          WISE, JAMES               40.0     3.25     $130.00
```

Figure 4-2 Printer Output

MULTIPLICATION

The following example illustrates the basic use of the Multiply statement.



```
MULTIPLY RATE-CD BY HOURS-CD GIVING TOTAL-PAY-PR.
```

Figure 4-3 Example of MULTIPLY Statement

In the MULTIPLY statement illustrated in Figure 4-3, the RATE-CD field is multiplied by the HOURS-CD field and the answer is placed in the area referenced by TOTAL-PAY-PR.

The format of the Multiply statement is illustrated below.

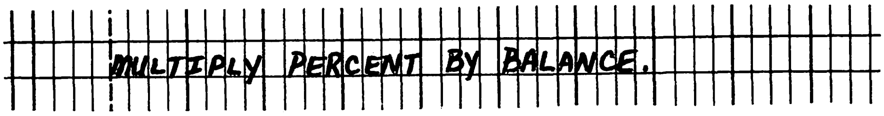
```
         ( identifier-1 )       ( identifier-2 )
MULTIPLY <              >   BY  <              >[GIVING identifier-3]
         ( literal-1    )       ( literal-2    )

     [ROUNDED]    [ON SIZE ERROR imperative-statement]
```

Figure 4-4 Format Notation – MULTIPLY Statement

The Multiply statement multiplies two numeric data items and sets the value of identifier-2 or identifier-3 equal to the product.

When the GIVING option is used, identifier-1 and identifier-2 are multiplied, the answer developed in a work area, and the answer is then moved to identifier-3. If a literal is used in the identifier-2 position, the GIVING option must be used.

When the GIVING option is omitted, the second operand must be an identifier and the product replaces the value in the field specified by the identifier. For example, the following Multiply statement would result in the product being placed at BALANCE.

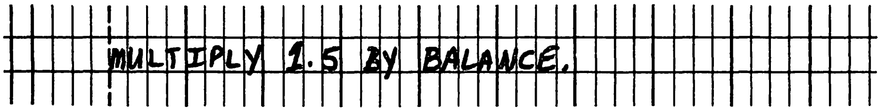


```
MULTIPLY PERCENT BY BALANCE.
```

Figure 4-5 Example of MULTIPLY Statement

In the MULTIPLY Statement illustrated above, PERCENT is multiplied by BALANCE and the answer is stored in the area referenced by BALANCE.

Numeric literals may also be specified for either the multiplier or the multiplicand when the GIVING Option is used. The following example illustrates the use of a numeric literal in a Multiply Statement.



```
MULTIPLY 1.5 BY BALANCE.
```

Figure 4-6 MULTIPLY Statement using a Numeric Literal

In this example, 1.5 is multiplied by BALANCE and the answer is stored in the area referenced by BALANCE.

Rounded Option

If, after decimal point alignment, the number of places to the right of the decimal point in the calculated result is greater than the number of places associated with the identifier whose value is to be set equal to the calculated result, truncation occurs unless the ROUNDED option has been specified. (Truncation refers to the dropping of digits from an answer).

When the ROUNDED option is specified, the least significant digit of the desired result is increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

The following chart illustrates the relationship between a calculated result and the value stored in an item that is to receive the calculated result.

	Item to Receive Calculated Result		
Calculated Result	PICTURE	Value After Rounding	Value After Truncating
12.36	99V9	12.4	12.3
8.432	9V9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	99V	66	65
.0055	V999	.006	.005

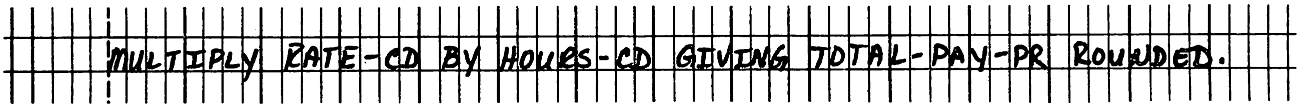
Figure 4-7 Chart Illustrating Values before and after Rounding

To round off an answer using COBOL the word ROUNDED is included in the Multiply Statement. In the sample problem developed in the text a four digit Rate Field expressed in dollars and cents is to be multiplied by a three digit Hours Field expressed in terms of Hours and tenths of an Hour. The answer is to be rounded off and expressed in dollars and cents.

```
$ 10.00  Rate Field
  40.0   Hours Field
400.000
   ↑
   └─Decimal Position
```

NOTE: Answer is to be rounded off and printed as $400.00

The following example illustrates the use of a Multiplication Statement with the ROUNDED Option.



```
MULTIPLY RATE-CD BY HOURS-CD GIVING TOTAL-PAY-PR ROUNDED.
```

Figure 4-8 Example of Multiply Statement with Rounded Option

In the illustration of the MULTIPLY Statement above, RATE–CD is multiplied by HOURS–CD and the answer is rounded off and stored in the area referenced by TOTAL–PAY–PR.

Size Error Option

Whenever the number of integral places in the calculated result exceeds the number of integral places specified for the resultant data-name, a SIZE ERROR condition exists. The following example illustrates a Size Error Condition.

```
03  TOTAL-PAY-PR              PICTURE $ZZZ.99.
```

$ 25.00 Rate Field
40.0 Hours Field
$1,000.000
↑Decimal Point

NOTE: Answer developed after rounding contains 6 digits. Picture is set up to contain a maximum of 5 digits. When the fields specified are multiplied, a SIZE ERROR condition will result.

Figure 4-9 Illustration of a Picture Clause and Related Field Causing Size Error

In the example above, the Picture Clause is specified for a field 5 digits in length, but the answer developed contains 6 digits. Thus, a Size Error results.

If the SIZE ERROR Option has been specified and a size error condition arises, the value of the resultant identifier is not altered and the series of imperative statements specified for the condition is executed. The following example illustrates the use of the SIZE ERROR Option.

```
03
04    MULTIPLY RATE-CD BY HOURS-CD GIVING TOTAL-PAY-PR ROUNDED,
05        ON SIZE ERROR MOVE '*** EXCESSIVE HOURS OR PAY RATE ***'
06        TO MSG-PR.
07
```

Figure 4-10 MULTIPLY Statement with Rounded and Size Error Options

Note that on a Size Error the instruction to "MOVE '***EXCESSIVE HOURS OR PAY RATE***' TO MSG-PR" is executed. The value moved is an example of a non-numerical literal.

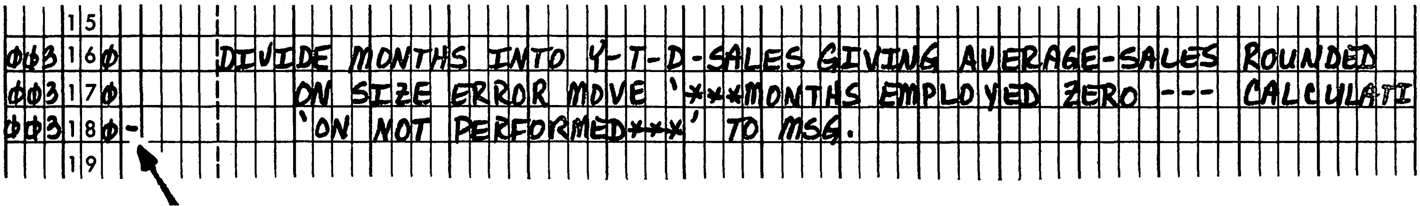
A non-numeric literal is a constant that may be used in certain COBOL statements. A non-numeric literal must be bounded by single quotation marks and may consist of any combination of characters except quotation marks. All spaces enclosed by the quotation marks are included as part of the literal. A non-numeric literal may not exceed 120 characters in length.

If the SIZE ERROR Option has not been specified and a size error condition arises, no assumption should be made about the final result. However, the program flow is not interrupted.

DIVIDE

The Divide statement divides one numeric data item into another and stores the quotient in the specified field.

The following example illustrates the use of the Divide statement.



```
   15
003160     DIVIDE MONTHS INTO Y-T-D-SALES GIVING AVERAGE-SALES ROUNDED
003170         ON SIZE ERROR MOVE '***MONTHS EMPLOYED ZERO --- CALCULATI
003180-        'ON NOT PERFORMED***' TO MSG.
   19
```

Continuation Punch

Figure 4-11 Example of the Divide Statement

In this example MONTHS is divided into Y-T-D SALES and the answer is stored in the area referenced by AVERAGE-SALES.

The COBOL statement in Figure 4-11 also illustrates entries required when a non-numeric literal is continued on an additional line of coding.

When a non-numeric literal occupies more than one line of a coding sheet, the following rules apply:

1. A hyphen is placed in column 7 of the continuation line.
2. A quotation mark is placed in Area B preceding the continuation of the literal.
3. All spaces at the end of the continued line and any spaces following the quotation mark in the continuation line and preceding the final quotation mark of the literal are considered part of the literal.

The format of the Divide statement is given below.

```
         { identifier-1 }
DIVIDE   {              }  INTO identifier-2  [ROUNDED]
         { literal-1    }

      [ON SIZE ERROR imperative-statement]
```

OR

```
         { identifier-1 } { INTC } { identifier-2 }
DIVIDE   {              } {      } {              }  GIVING identifier-3
         { literal-1    } { BY   } { literal-2    }

      [ROUNDED] [REMAINDER identifier-4]

      [ON SIZE ERROR imperative-statement]
```

Figure 4-12 Format Notation – DIVIDE Statement

Note that both the ROUNDED option and the ON SIZE ERROR option may be used with the Divide statement, as with all arithmetic statements.

With division, if the divisor is zero, a SIZE ERROR condition results.

When the GIVING option is omitted and the second operand is an identifier, division results in the value stored in the field specified by identifier-2 being set equal to the quotient.

```
DIVIDE TIME INTO HOURS.
```

Figure 4-13 Example of the DIVIDE Statement

In the example above TIME is divided into HOURS and the answer is stored in the area referenced by HOURS.

A numeric literal may be specified as either the divisor or the dividend when the GIVING option is used. The following example illustrates the use of a Divide statement with a numeric literal specified in the divisor.

```
DIVIDE 12 INTO MONTHS GIVING AVERAGE.
```

Figure 4-14 Example of the DIVIDE Statement using a Numeric Literal

In this example 12 is divided into MONTHS and the answer is stored in the area referenced by AVERAGE.

SAMPLE PROBLEM

The Input, Output, Flowchart and Coding for a sample problem involving multiplication is illustrated on the following pages.

Input

Input for the sample problem consists of Employee Time Cards containing an Employee Number Field, a Name Field, an Hours Field, and a Rate Field.

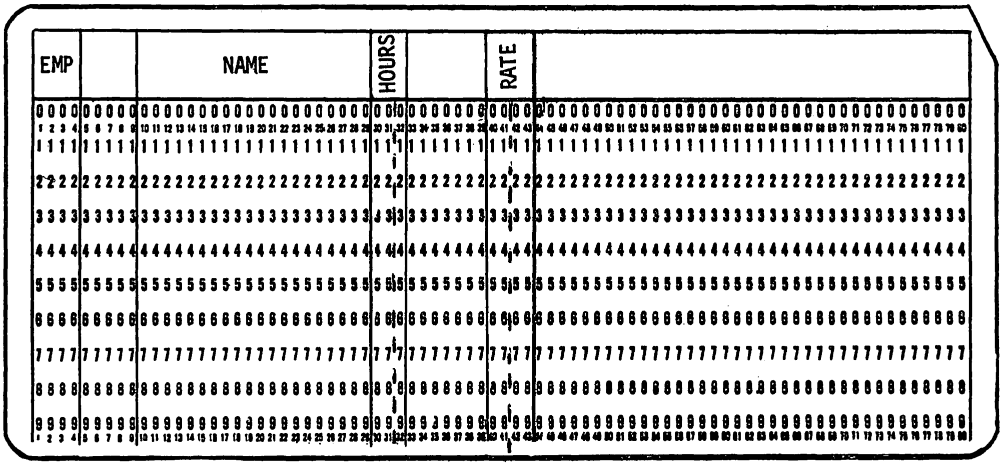
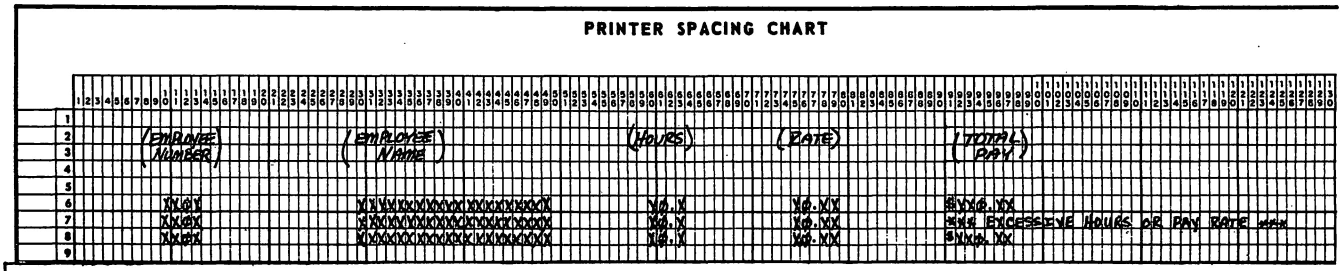


Figure 4-15 Input for Sample Program

Output

Output is to consist of a printed report in which each of the fields are printed. In addition, Total Pay is to be calculated by multiplying Hours times Rate. No pay should exceed $999.99. If the calculated pay exceeds $999.99, a message is printed on the report. Note in the sample COBOL program the technique for printing the message, and how the area in which the message is to be printed is specified as a Level 02 entry. See Line 003110 of the COBOL Program.



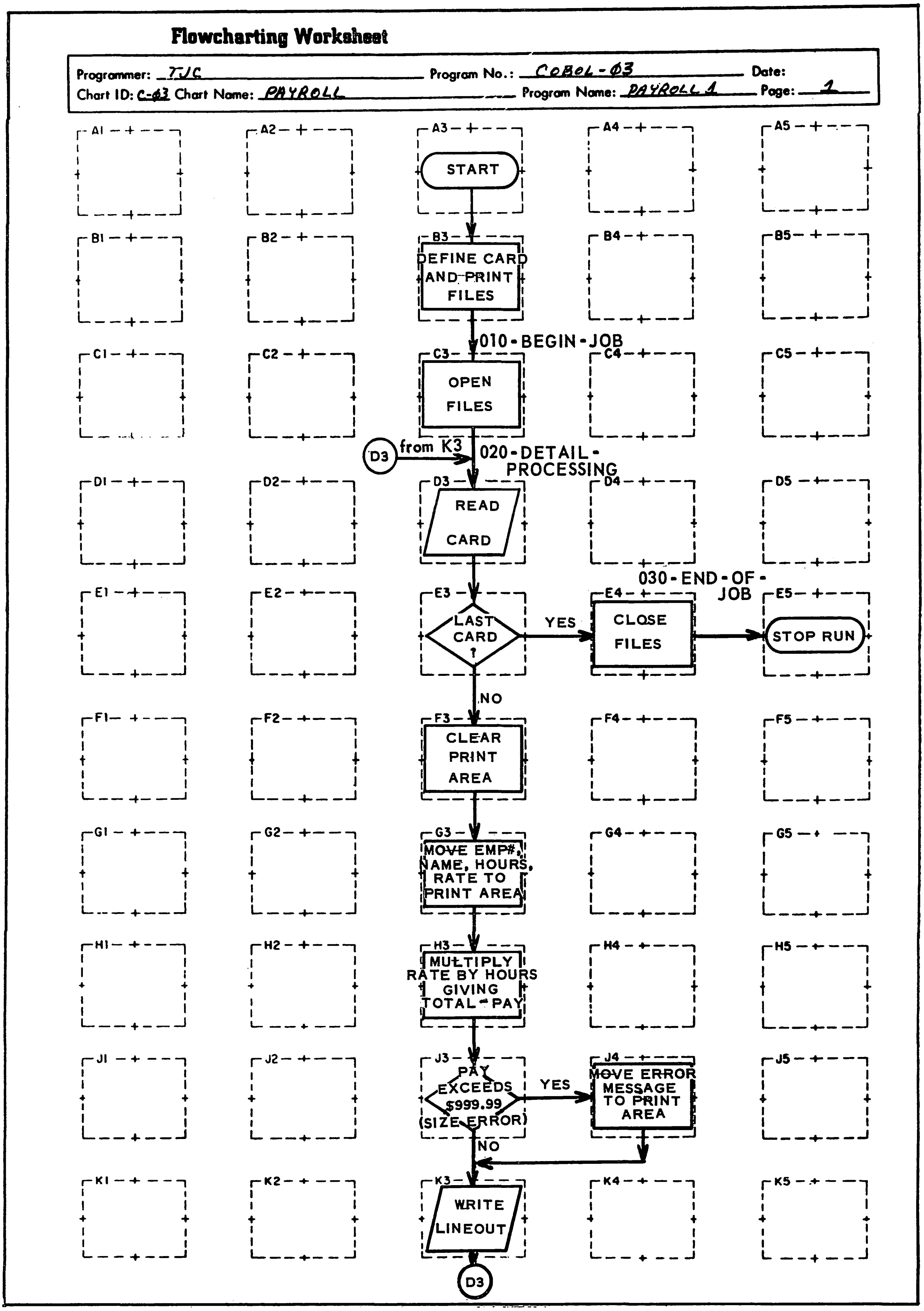
PRINTER SPACING CHART

```
2     (EMPLOYEE     (EMPLOYEE                (HOURS)    (RATE)     (TOTAL
3      NUMBER)        NAME )                                         PAY )
4
5
6       XXØX        XXXXXXXXXXXXXXXXXXXX       XØ.X       XØ.XX      $XXØ.XX
7       XXØX        XXXXXXXXXXXXXXXXXXXX       XØ.X       XØ.XX      *** EXCESSIVE HOURS OR PAY RATE ***
8       XXØX        XXXXXXXXXXXXXXXXXXXX       XØ.X       XØ.XX      $XXØ.XX
9
```

```
1000             ABBOT, ROY                 40.0         2.00       $ 80.00
1010             DEROUCHEY, MARY            39.5         2.00       $ 79.00
1020             DUNN, THOMAS               38.0         1.95       $ 74.10
5213             STACY, LEROY               79.8        32.00       *** EXCESSIVE HOURS OR PAY RATE ***
8913             WISE, JAMES                40.0         3.25       $130.00
```

Figure 4-16 Output for Sample Program

FLOWCHART



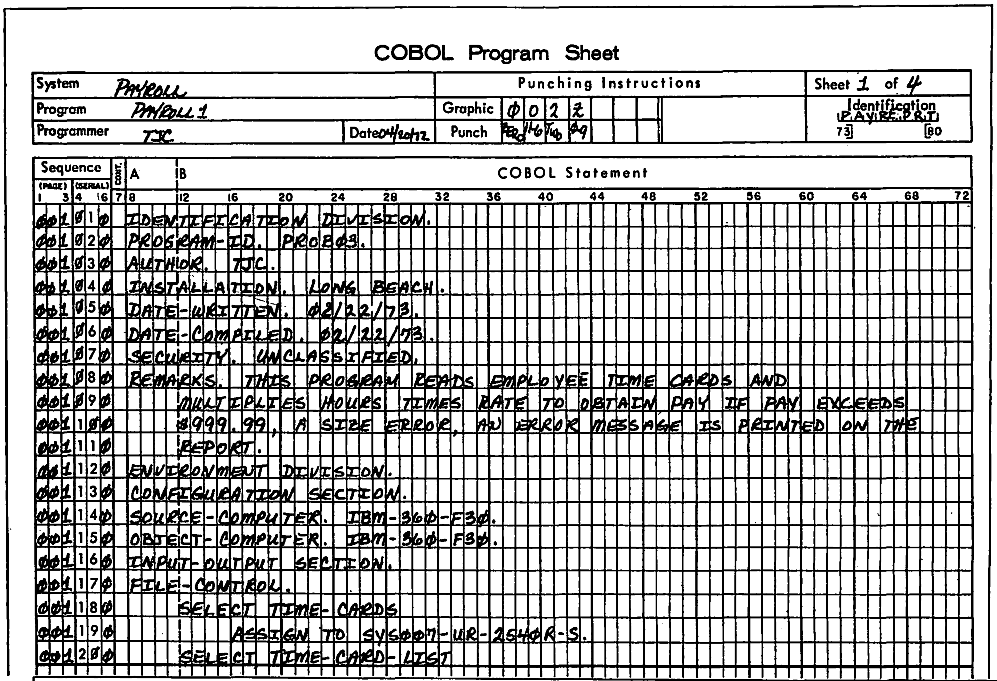
Flowcharting Worksheet

Programmer: TJC Program No.: COBOL-03 Date:
Chart ID: C-03 Chart Name: PAYROLL Program Name: PAYROLL 1 Page: 1

A1
A2
A3
A4
A5
START
B1
B2
B3
B4
B5
DEFINE CARD AND PRINT FILES
010-BEGIN-JOB
C1
C2
C3
C4
C5
OPEN FILES
D3
from K3
020-DETAIL-PROCESSING
D1
D2
D3
D4
D5
READ CARD
030-END-OF-JOB
E1
E2
E3
E4
E5
LAST CARD ?
YES
CLOSE FILES
STOP RUN
NO
F1
F2
F3
F4
F5
CLEAR PRINT AREA
G1
G2
G3
G4
G5
MOVE EMP#, NAME, HOURS, RATE TO PRINT AREA
H1
H2
H3
H4
H5
MULTIPLY RATE BY HOURS GIVING TOTAL-PAY
J1
J2
J3
J4
J5
PAY EXCEEDS $999.99 (SIZE ERROR)
YES
MOVE ERROR MESSAGE TO PRINT AREA
NO
K1
K2
K3
K4
K5
WRITE LINEOUT
D3

Figure 4-17 Program Flowchart

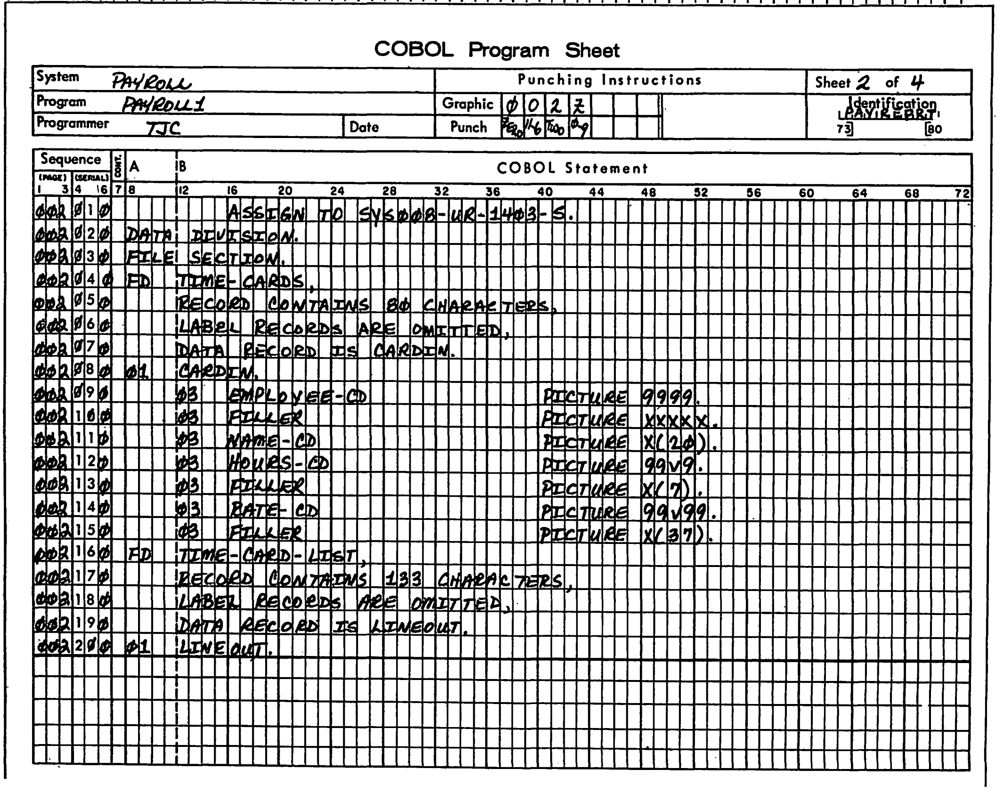
SOURCE CODING



COBOL Program Sheet

System PAYROLL
Program PAYROLL1
Programmer TJC
Date 04/20/72
Punching Instructions
Graphic: 0 | O | 2 | Z
Punch: Zero | 11-6 | Two | 0-9
Sheet 1 of 4
Identification PAYREPRT (73–80)

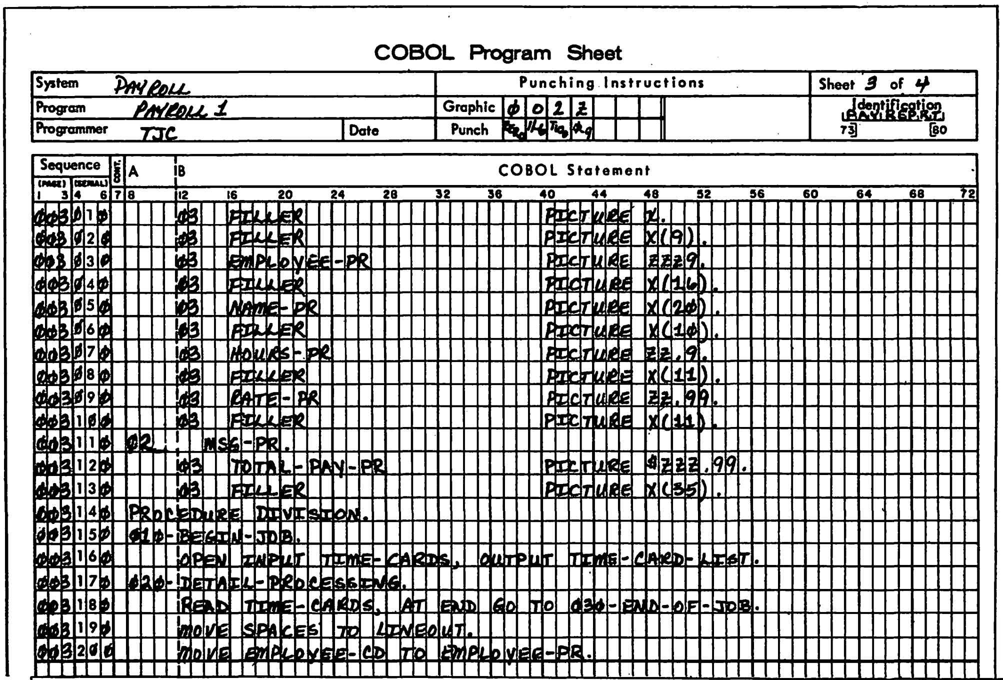
```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. PRO803.
001030 AUTHOR. TJC.
001040 INSTALLATION. LONG BEACH.
001050 DATE-WRITTEN. 02/22/73.
001060 DATE-COMPILED. 02/22/73.
001070 SECURITY. UNCLASSIFIED.
001080 REMARKS. THIS PROGRAM READS EMPLOYEE TIME CARDS AND
001090         MULTIPLIES HOURS TIMES RATE TO OBTAIN PAY IF PAY EXCEEDS
001100         $999.99, A SIZE ERROR, AN ERROR MESSAGE IS PRINTED ON THE
001110         REPORT.
001120 ENVIRONMENT DIVISION.
001130 CONFIGURATION SECTION.
001140 SOURCE-COMPUTER. IBM-360-F30.
001150 OBJECT-COMPUTER. IBM-360-F30.
001160 INPUT-OUTPUT SECTION.
001170 FILE-CONTROL.
001180         SELECT TIME-CARDS
001190             ASSIGN TO SYS007-UR-2540R-S.
001200         SELECT TIME-CARD-LIST
```



COBOL Program Sheet

System PAYROLL
Program PAYROLL1
Programmer TJC
Date
Punching Instructions
Graphic: 0 | O | 2 | Z
Punch: Zero | 11-6 | Two | 0-9
Sheet 2 of 4
Identification PAYREPRT (73–80)

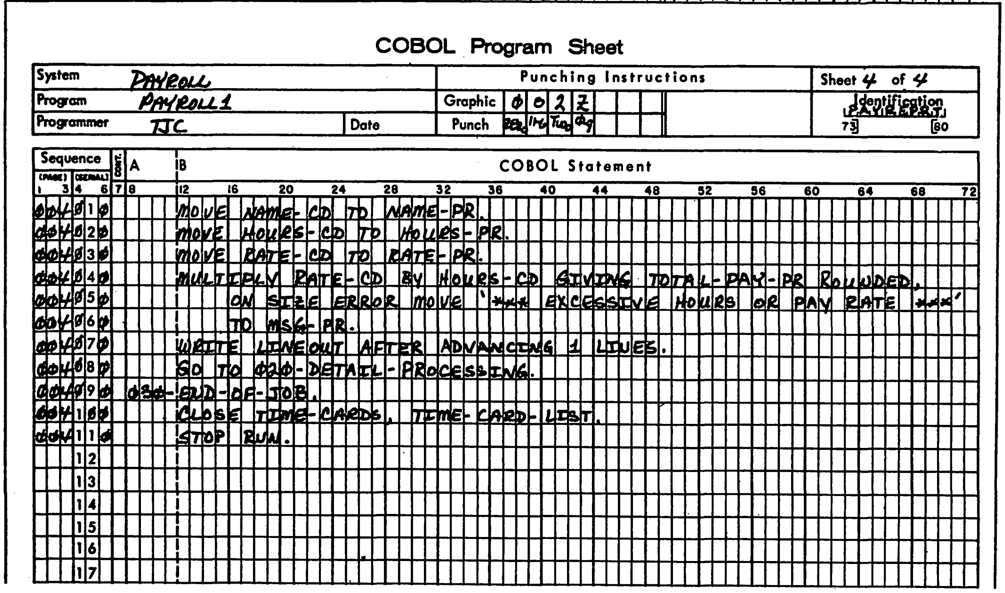
```
002010             ASSIGN TO SYS008-UR-1403-S.
002020 DATA DIVISION.
002030 FILE SECTION.
002040 FD  TIME-CARDS,
002050         RECORD CONTAINS 80 CHARACTERS,
002060         LABEL RECORDS ARE OMITTED,
002070         DATA RECORD IS CARDIN.
002080 01  CARDIN.
002090         03  EMPLOYEE-CD             PICTURE 9999.
002100         03  FILLER                  PICTURE XXXXX.
002110         03  NAME-CD                 PICTURE X(20).
002120         03  HOURS-CD                PICTURE 99V9.
002130         03  FILLER                  PICTURE X(7).
002140         03  RATE-CD                 PICTURE 99V99.
002150         03  FILLER                  PICTURE X(37).
002160 FD  TIME-CARD-LIST,
002170         RECORD CONTAINS 133 CHARACTERS,
002180         LABEL RECORDS ARE OMITTED,
002190         DATA RECORD IS LINEOUT.
002200 01  LINEOUT.
```



COBOL Program Sheet

System PAYROLL
Program PAYROLL 1
Programmer TJC
Date
Punching Instructions
Graphic 0 O 2 Z
Punch Zero 11-6 Two 0-9
Sheet 3 of 4
Identification PAYREPRT 73 80

```
003010     03  FILLER                  PICTURE X.
003020     03  FILLER                  PICTURE X(9).
003030     03  EMPLOYEE-PR             PICTURE ZZZ9.
003040     03  FILLER                  PICTURE X(16).
003050     03  NAME-PR                 PICTURE X(20).
003060     03  FILLER                  PICTURE X(10).
003070     03  HOURS-PR                PICTURE ZZ.9.
003080     03  FILLER                  PICTURE X(11).
003090     03  RATE-PR                 PICTURE ZZ.99.
003100     03  FILLER                  PICTURE X(11).
003110 02    MSG-PR.
003120     03   TOTAL-PAY-PR           PICTURE $ZZZ.99.
003130     03  FILLER                  PICTURE X(35).
003140 PROCEDURE DIVISION.
003150 010-BEGIN-JOB.
003160     OPEN INPUT TIME-CARDS, OUTPUT TIME-CARD-LIST.
003170 020-DETAIL-PROCESSING.
003180     READ TIME-CARDS, AT END GO TO 030-END-OF-JOB.
003190     MOVE SPACES TO LINEOUT.
003200     MOVE EMPLOYEE-CD TO EMPLOYEE-PR.
```



COBOL Program Sheet

System PAYROLL
Program PAYROLL1
Programmer TJC
Date
Punching Instructions
Graphic 0 O 2 Z
Punch Zero 11-6 Two 0-9
Sheet 4 of 4
Identification PAYREPRT 73 80

```
004010     MOVE NAME-CD TO NAME-PR.
004020     MOVE HOURS-CD TO HOURS-PR.
004030     MOVE RATE-CD TO RATE-PR.
004040     MULTIPLY RATE-CD BY HOURS-CD GIVING TOTAL-PAY-PR ROUNDED,
004050             ON SIZE ERROR MOVE '*** EXCESSIVE HOURS OR PAY RATE ***'
004060             TO MSG-PR.
004070     WRITE LINEOUT AFTER ADVANCING 1 LINES.
004080     GO TO 020-DETAIL-PROCESSING.
004090 030-END-OF-JOB.
004100     CLOSE TIME-CARDS, TIME-CARD-LIST.
004110     STOP RUN.
   12
   13
   14
   15
   16
   17
```

SOURCE LISTING

```
      1                                          IBM DOS AMERICAN NATIONAL STANDARD COBOL         CBF CL3-4              02/22/73

00001   001010 IDENTIFICATION DIVISION.                                         PAYREPRT
00002   001020 PROGRAM-ID. PROB03.                                              PAYREPRT
00003   001030 AUTHOR. TJC.                                                     PAYREPRT
00004   001040 INSTALLATION. LONG BEACH.                                        PAYREPRT
00005   001050 DATE-WRITTEN. 02/22/73.                                          PAYREPRT
00006   001060 DATE-COMPILED. 02/22/73.                                         PAYREPRT
00007   001070 SECURITY. UNCLASSIFIED.                                          PAYREPRT
00008   001080 REMARKS. THIS PROGRAM READS EMPLOYEE TIME CARDS AND              PAYREPRT
00009   001090     MULTIPLIES HOURS TIMES RATE TO OBTAIN PAY IF PAY EXCEEDS     PAYREPRT
00010   001100     $999.99, A SIZE ERROR, AN ERROR MESSAGE IS PRINTED ON THE    PAYREPRT
00011   001110     REPORT.                                                      PAYREPRT
00012   001120 ENVIRONMENT DIVISION.                                            PAYREPRT
00013   001130 CONFIGURATION SECTION.                                           PAYREPRT
00014   001140 SOURCE-COMPUTER. IBM-360-F30.                                    PAYREPRT
00015   001150 OBJECT-COMPUTER. IBM-360-F30.                                    PAYREPRT
00016   001160 INPUT-OUTPUT SECTION.                                            PAYREPRT
00017   001170 FILE-CONTROL.                                                    PAYREPRT
00018   001180     SELECT TIME-CARDS                                            PAYREPRT
00019   001190         ASSIGN TO SYS007-UR-2540R-S.                             PAYREPRT
00020   001200     SELECT TIME-CARD-LIST                                        PAYREPRT
00021   002010         ASSIGN TO SYS008-UR-1403-S.                              PAYREPRT
00022   002020 DATA DIVISION.                                                   PAYREPRT
00023   002030 FILE SECTION.                                                    PAYREPRT
00024   002040 FD  TIME-CARDS,                                                  PAYREPRT
00025   002050     RECORD CONTAINS 80 CHARACTERS,                               PAYREPRT
00026   002060     LABEL RECORDS ARE OMITTED,                                   PAYREPRT
00027   002070     DATA RECORD IS CARDIN.                                       PAYREPRT
00028   002080 01  CARDIN.                                                      PAYREPRT
00029   002090     03  EMPLOYEE-CD             PICTURE 9999.                    PAYREPRT
00030   002100     03  FILLER                  PICTURE XXXXX.                   PAYREPRT
00031   002110     03  NAME-CD                 PICTURE X(20).                   PAYREPRT
00032   002120     03  HOURS-CD                PICTURE 99V9.                    PAYREPRT
00033   002130     03  FILLER                  PICTURE X(7).                    PAYREPRT
00034   002140     03  RATE-CD                 PICTURE 99V99.                   PAYREPRT
00035   002150     03  FILLER                  PICTURE X(37).                   PAYREPRT
00036   002160 FD  TIME-CARD-LIST,                                              PAYREPRT
00037   002170     RECORD CONTAINS 133 CHARACTERS,                              PAYREPRT
00038   002180     LABEL RECORDS ARE OMITTED,                                   PAYREPRT
00039   002190     DATA RECORD IS LINEOUT.                                      PAYREPRT
00040   002200 01  LINEOUT.                                                     PAYREPRT
00041   003010     03  FILLER                  PICTURE X.                       PAYREPRT
00042   003020     03  FILLER                  PICTURE X(9).                    PAYREPRT
00043   003030     03  EMPLOYEE-PR             PICTURE ZZZ9.                    PAYREPRT
00044   003040     03  FILLER                  PICTURE X(16).                   PAYREPRT
00045   003050     03  NAME-PR                 PICTURE X(20).                   PAYREPRT
00046   003060     03  FILLER                  PICTURE X(10).                   PAYREPRT
00047   003070     03  HOURS-PR                PICTURE ZZ.9.                    PAYREPRT
00048   003080     03  FILLER                  PICTURE X(11).                   PAYREPRT
00049   003090     03  RATE-PR                 PICTURE ZZ.99.                   PAYREPRT
00050   003100     03  FILLER                  PICTURE X(11).                   PAYREPRT
00051   003110 02  MSG-PR.                                                      PAYREPRT
00052   003120     03  TOTAL-PAY-PR            PICTURE $ZZZ.99.                 PAYREPRT
00053   003130     03  FILLER                  PICTURE X(35).                   PAYREPRT
00054   003140 PROCEDURE DIVISION.                                              PAYREPRT
00055   003150 010-BEGIN-JOB.                                                   PAYREPRT
00056   003160     OPEN INPUT TIME-CARDS, OUTPUT TIME-CARD-LIST.                PAYREPRT
00057   003170 020-DETAIL-PROCESSING.                                           PAYREPRT
00058   003180     READ TIME-CARDS, AT END GO TO 030-END-OF-JOB.                PAYREPRT
00059   003190     MOVE SPACES TO LINEOUT.                                      PAYREPRT
00060   003200     MOVE EMPLOYEE-CD TO EMPLOYEE-PR.                             PAYREPRT
00061   004010     MOVE NAME-CD TO NAME-PR.                                     PAYREPRT
00062   004020     MOVE HOURS-CD TO HOURS-PR.                                   PAYREPRT
00063   004030     MOVE RATE-CD TO RATE-PR.                                     PAYREPRT
00064   004040     MULTIPLY RATE-CD BY HOURS-CD GIVING TOTAL-PAY-PR ROUNDED,    PAYREPRT
00065   004050         ON SIZE ERROR MOVE '*** EXCESSIVE HOURS OR PAY RATE ***' PAYREPRT
00066   004060         TO MSG-PR.                                               PAYREPRT
00067   004070     WRITE LINEOUT AFTER ADVANCING 1 LINES.                       PAYREPRT
00068   004080     GO TO 020-DETAIL-PROCESSING.                                 PAYREPRT
00069   004090 030-END-OF-JOB.                                                  PAYREPRT
00070   004100     CLOSE TIME-CARDS, TIME-CARD-LIST.                            PAYREPRT
00071   004110     STOP RUN.                                                    PAYREPRT
```

CHAPTER 4

PROGRAMMING ASSIGNMENT 1

INSTRUCTIONS

From the Sales Cards illustrated below a Sales Commission Report is to be prepared. The Commission is to be calculated by multiplying Current-Sales by the Commission Rate (percent). Round off so that the answer is expressed in terms of dollars and cents. The Commission Rate varies with each department.

It has been determined that no Commission Amount should exceed $9,999.99. If the calculated Commission Amount exceeds $9,999.99, through the use of the Size Error Option print the message "***EXCESSIVE COMMISSION***".

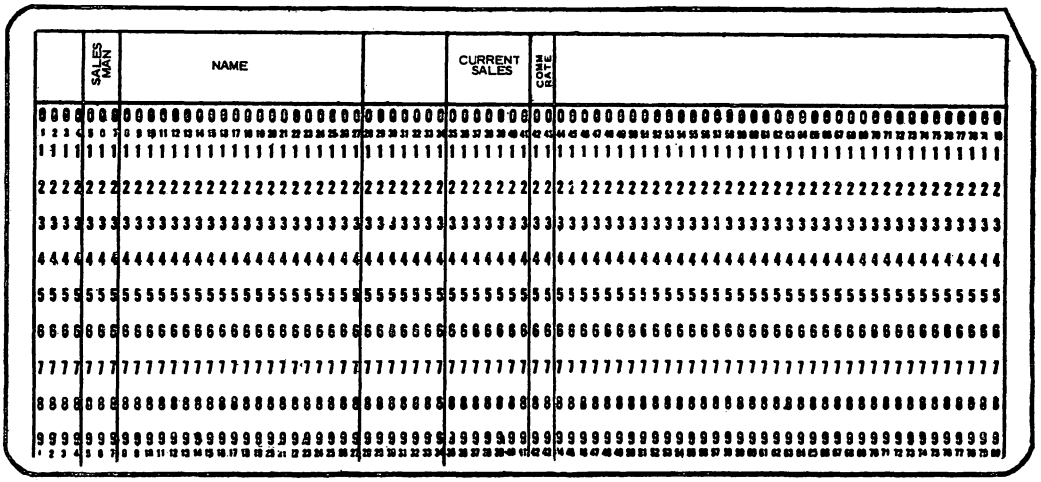
Edit the Current Sales field with a dollar sign, comma, and decimal point. Print a percent sign (%) to the right of the Commission Rate. Edit the Commission Amount with a floating dollar sign.

Cards are to be sorted by Salesman Number.

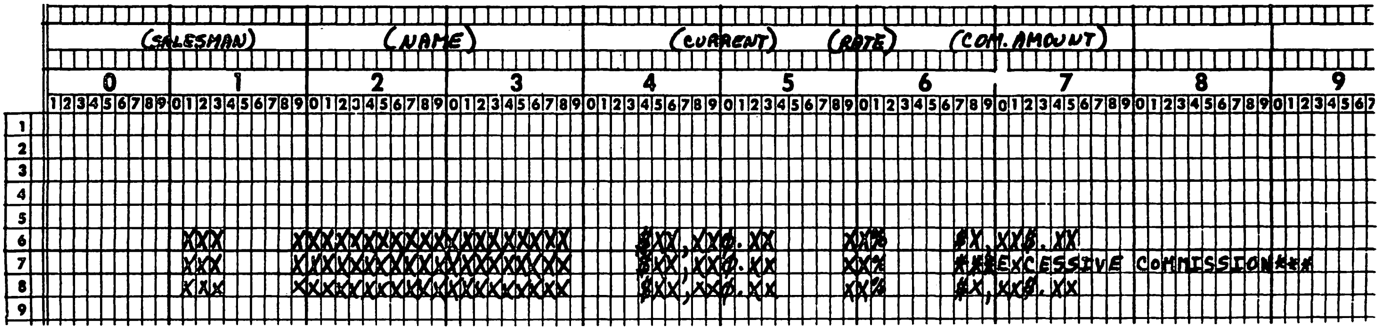
Prepare a flowchart and the COBOL program to produce the required report.

The test data is contained in the appendices.

INPUT: Sales Cards


SALES MAN
NAME
CURRENT SALES
COMM RATE

OUTPUT: Sales Commission Report


(SALESMAN)
(NAME)
(CURRENT)
(RATE)
(COM. AMOUNT)
XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX XX% $X,XX0.XX
XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX XX% ***EXCESSIVE COMMISSION***
XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX XX% $X,XX0.XX

CHAPTER 4

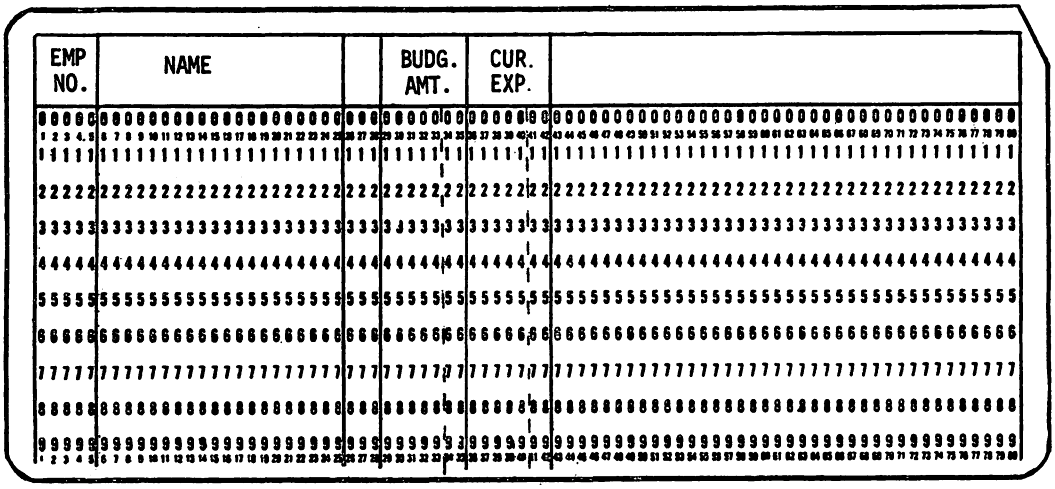
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

Write the COBOL program to produce a Budget Report. The input and output are illustrated below.

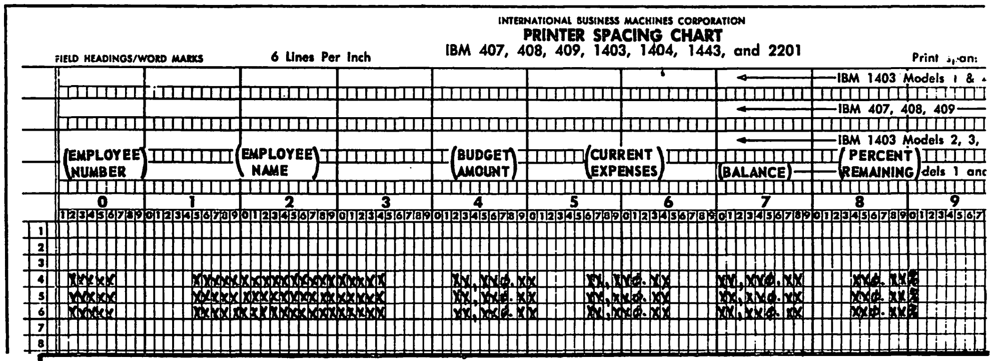
INPUT

The input to the program consists of a file of Budget Expense cards. They contain the Employee Number, the Employee Name, the Budget Amount, and the Current Expenses.


EMP NO.
NAME
BUDG. AMT.
CUR. EXP.

OUTPUT

The output from the program is a Budget Report containing the Employee Number, the Employee Name, the Budget Amount, the Current Expenses, the Balance in the Budget, and the Percentage of the Budget Remaining. The Balance is calculated by subtracting the Current Expenses from the Budget Amount. The Percentage is calculated by dividing the Balance by the Budget Amount, that is, Balance ÷ Amount = Percentage.


INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, and 2201
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
IBM 1403 Models 1 &
IBM 407, 408, 409
IBM 1403 Models 2, 3,
EMPLOYEE NUMBER
EMPLOYEE NAME
BUDGET AMOUNT
CURRENT EXPENSES
BALANCE
PERCENT REMAINING

```
63481      ANDERSON, J.        10,000.00     5,000.00     5,000.00      50.00%
68274      BARCLAY, B.         25,000.00    10,000.00    15,000.00      60.00%
67891      JAMIESON, B.        20,000.00          .00    20,000.00     100.00%
```

TEST DATA – ASSIGNMENT NO. 2

Emp. No. (col 1-5)	Name (col 6-25)	Budget Amount (col 29-35)	Expenses (col 36-42)
63481	Anderson, J.	1000000	0500000
68274	Barclay, B.	2500000	1000000
67891	Jamieson, B.	2000000	0000000
74310	Kiler, G.	4500000	1123400
89763	Milner, F.	1000000	0135896
99784	Ziepak, K.	3500000	2976509

CHAPTER 5

COMPUTE STATEMENT

FINAL TOTALS

COMPUTE STATEMENT

In many applications it is necessary to perform a series of arithmetic calculations. With the COMPUTE Verb, the programmer can use arithmetic expressions to specify one or a series of arithmetic operations. The following example illustrates the use of a COMPUTE Statement.

```
COMPUTE PROFIT = SALES - COST.
```

Figure 5-1 Example of the COMPUTE Statement

In the COMPUTE Statement illustrated above the calculation specified on the right side of the equal sign is performed and the answer is moved to the identifier specified on the left side of the equal sign. In this example, COST is subtracted from SALES and the answer is stored in the area referenced by PROFIT.

The COMPUTE Statement permits most arithmetic operations to be specified in a much more concise manner than is possible using the simple arithmetic verbs of ADD, SUBTRACT, MULTIPLY, and DIVIDE.

The format of the COMPUTE Statement is illustrated below:

```
                                      ( identifier-2          )
COMPUTE identifier-1 [ROUNDED] =     <  literal-1              >
                                      ( arithmetic-expression )

     [ON SIZE ERROR imperative-statement]
```

Figure 5-2 COMPUTE Statement – Format Notation

The COMPUTE Statement assigns to a data item the value of a numeric data item, literal, or arithmetic expression. The identifier specified to the left of the equal sign must be an elementary report, internal decimal, or external decimal item.

The Arithmetic expression specified in the COMPUTE Statement may consist of "Arithmetic Operators", identifiers, and/or literals representing items on which arithmetic operations may be performed.

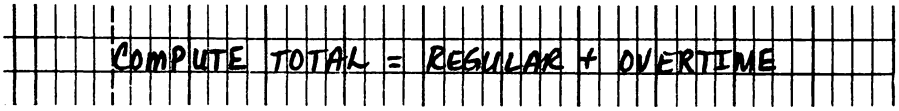
The following five " Arithmetic Operators " may be used in arithmetic expressions.

Operator	Arithmetic Operation
+	Addition
−	Subtraction
°	Multiplication
/	Division
°°	Exponentiation

Figure 5-3 Arithmetic Operators

The following examples illustrate the use of Arithmetic Operations in the COMPUTE Statement. Note the space before and after the equal sign and each of the arithmetic operators. An Arithmetic Operator must always be preceded and followed by a space.

Example 1: Add

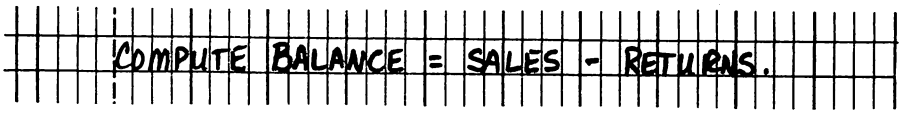


```
COMPUTE TOTAL = REGULAR + OVERTIME
```

Figure 5-4 Example 1 – Add

In Example 1, REGULAR is added to OVERTIME and the answer is stored in the area referenced by TOTAL.

Example 2: Subtract



```
COMPUTE BALANCE = SALES - RETURNS.
```

Figure 5-5 Example 2 – Subtract

In Example 2, RETURNS is subtracted from SALES and the answer is stored in the area referenced by BALANCE.

Example 3: Multiply

```
COMPUTE PAY = HOURS * RATE.
```

Figure 5-6 Example 3 – Multiply

In Example 3, HOURS is multiplied by RATE and the answer is stored in the area referenced by PAY.

Example 4: Divide

```
COMPUTE AVERAGE = PAY / MONTHS.
```

Figure 5-7 Example 4 – Divide

In Example 4, MONTHS is divided into PAY and the answer is stored in the area referenced by AVERAGE.

Example 5: Addition, Subtraction

```
COMPUTE NET-PAY ROUNDED = REGULAR + OVERTIME - DEDUCTIONS.
```

Figure 5-8 Example 5 – Addition, Subtraction

In Example 5, REGULAR is added to OVERTIME, the area referenced by DEDUCTIONS is subtracted, and the answer is stored in the area referenced by NET-PAY.

Example 6: Exponentiation

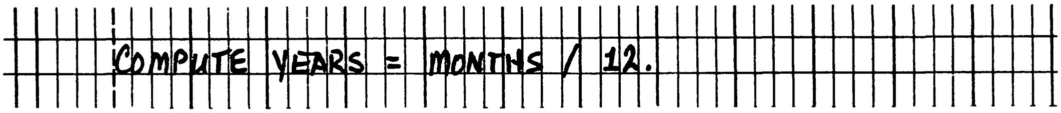
Exponentiation means raising to a power; thus, A^2 means A times A. This could be written in COBOL as A ** 2. The following example Illustrates a COMPUTE Statement with exponentiation.

```
COMPUTE VOLUME = 4 / 3 * PI * R ** 3.
```

Figure 5-9 Example 6 – Exponentiation

Example 7: Numeric Literal

Numeric Literals may be specified within the COMPUTE Statement. The following example illustrates the use of a Numeric Literal.



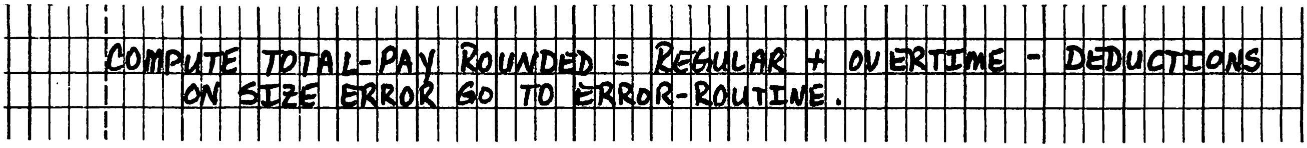
```
COMPUTE YEARS = MONTHS / 12.
```

Figure 5-10 Example 7 – Use of a Numeric Literal

In the Example above, 12 is divided into MONTHS and the answer is stored in the area referenced by YEARS.

Example 8: Rounded, Size Error Options

The following example illustrates the COMPUTE Statement incorporating the ROUNDED and SIZE ERROR Options.



```
COMPUTE TOTAL-PAY ROUNDED = REGULAR + OVERTIME - DEDUCTIONS
    ON SIZE ERROR GO TO ERROR-ROUTINE.
```

Figure 5-11 Rounded and Size Error Options

The ON SIZE ERROR Option applies only to the final result and not to any of the intermediate results.

Evaluating Arithmetic Expressions

The way in which an arithmetic expression is to be evaluated can be specified by parentheses. Thus, the expression A * B + C might be considered ambiguous. Does the programmer mean (A * B) + C, or does he mean A * (B + C)? In COBOL the programmer may use pairs of parentheses in order to describe exactly the way in which a computation is to be performed.

If parentheses are NOT written to specify the order of computation, COBOL will evaluate an arithmetic expression using the following rules:

1. All exponentiation is performed first.
2. Then, multiplication and division are performed.
3. Finally, addition and subtraction are performed.
4. In each of the three above steps, computation starts at the left of the expression and proceeds to the right. Thus, A * B / C is computed as (A * B) / C, and A / B * C is computed as (A / B) * C.
5. When parentheses are present, computation begins with the innermost set and proceeds to the outermost. Items grouped in parentheses will be evaluated in accordance with the above rules, and the result will then be treated as if the parentheses were removed.

The following examples illustrate the use of the COMPUTE Statement to compute depreciation. Depreciation is calculated by subtracting SALVAGE from COST and dividing the answer from this calculation by LIFE. Example 1 illustrates an INCORRECT method for writing the COMPUTE Statement. Example 2 illustrates the correct method

Example 1: Incorrect Method

```
COMPUTE DEPRECIATION-HOLD = COST-CD - SALVAGE-CD / LIFE-CD
```

Figure 5-12 Incorrect use of the COMPUTE Statement

In example 1, because of the hierarchy of operations as they are performed with COBOL, the division operation would be performed first (dividing LIFE-CD into SALVAGE-CD) and the subtraction operation would then be performed resulting in an erroneous answer.

The correct use of this Compute Statement is illustrated below.

Example 2: Correct Method

```
COMPUTE DEPRECIATION-HOLD = (COST-CD - SALVAGE-CD) / LIFE-CD
```

Figure 5-13 Correct use of the COMPUTE Statement

When the instruction illustrated above is executed, the calculations specified within the parentheses are performed, and the answer obtained would then be divided producing the correct answer.

The example below illustrates an additional use of the parentheses in specifying the hierarchy of operations. Note the use of the double parentheses.

```
COMPUTE AMOUNT = PRINCIPLE * ((1 + RATE) ** YEARS)
```

Figure 5-14 Use of the parentheses in the COMPUTE Statement

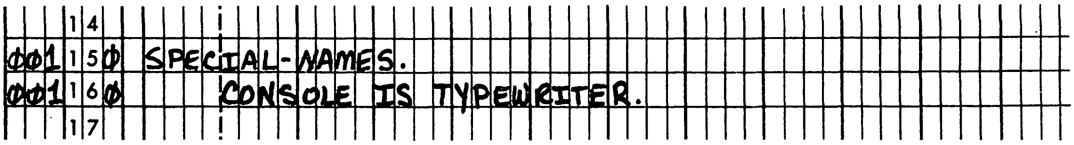
In this example the calculation (1 + RATE) would be performed first, this answer would be raised to the power specified by YEARS, and this answer multiplied by PRINCIPLE.

DISPLAY STATEMENT

To print brief messages on the console typewriter, the printer, or card punch the DISPLAY statement may be used. This statement is often used to give instructions to the operator concerning error conditions that might occur during execution of a program or to print "end of job" messages, etc.

In order to print a message on the console typewriter, an entry must be made in the Special-Names paragraph of the Configuration Section as well as the DISPLAY statement itself. The entries placed in the Special-Names Paragraph in the sample program are illustrated below.

EXAMPLE



```
001150 SPECIAL-NAMES.
001160     CONSOLE IS TYPEWRITER.
```

Figure 5-15 Special-Names Paragraph

The general format of the Special-Names Paragraph is illustrated below.

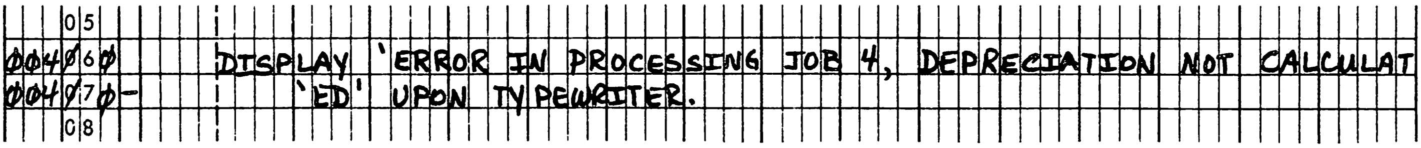
```
SPECIAL-NAMES.
     [function-name-1 IS mnemonic-name]...
     [function-name-2 [IS mnemonic-name]
        { CN STATUS IS condition-name-1
        { OFF STATUS IS condition-name-2
                     [OFF STATUS IS condition-name-2] }
                                                      }   ]...
                     [CN STATUS IS condition-name-1]  }
     [CURRENCY SIGN IS literal] [DECIMAL-POINT IS COMMA].
```

Figure 5-16 Special-Names Paragraph – Format Notation

The portion of the general format illustrated above which is used with the Display statement is the "function-name-1 IS mnemonic-name" portion. The "function-name-1" is one of a set of special names which describe a particular device or function to be performed. In the example in Figure 5-15, the function-name is the word CONSOLE, which must be used to indicate that the computer console typewriter is to be used for the output message. The mnemonic-name is a programmer-chosen word which is used in the DISPLAY statement to reference the typewriter console. In the example in Figure 5-15, it can be seen that the word chosen is TYPEWRITER.

When the DISPLAY statement is used, the mnemonic-name is included in the statement to indicate which device is to be used for the output. The statement used in the sample program is illustrated below.

EXAMPLE



```
004060    DISPLAY 'ERROR IN PROCESSING JOB 4, DEPRECIATION NOT CALCULAT
004070-           'ED' UPON TYPEWRITER.
```

Figure 5-17 DISPLAY Statement

The format of the Display statement is illustrated below.

```
            (literal-1    )  [literal-2    ]
DISPLAY     {             }  [             ] ...
            (identifier-1 )  [identifier-2 ]

     [UPON   mnemonic-name   ]
```

Figure 5-18 DISPLAY Statement — Format Notation

The function of the DISPLAY statement is to write on a low volume device. When the "UPON" clause is included in the Display statement, the device is defined by the mnemonic-name which is included in the statement. If the UPON clause is not included, the output will be written either on the typewriter console, or upon the system printer, depending upon the computer being used.

The maximum length of the message printed will normally be the maximum length record which can be printed on the particular device. On an IBM typewriter console, for example, the maximum length record is 72 characters.

FINAL TOTALS

Most business applications require the taking of "final totals" of calculations that have been performed. The printing of final totals requires an expansion of the basic flowchart logic which has been developed in previous chapters and also requires an understanding of the WORKING-STORAGE SECTION of a COBOL program.

The following flowchart illustrates the basic logic for the taking of a final total.

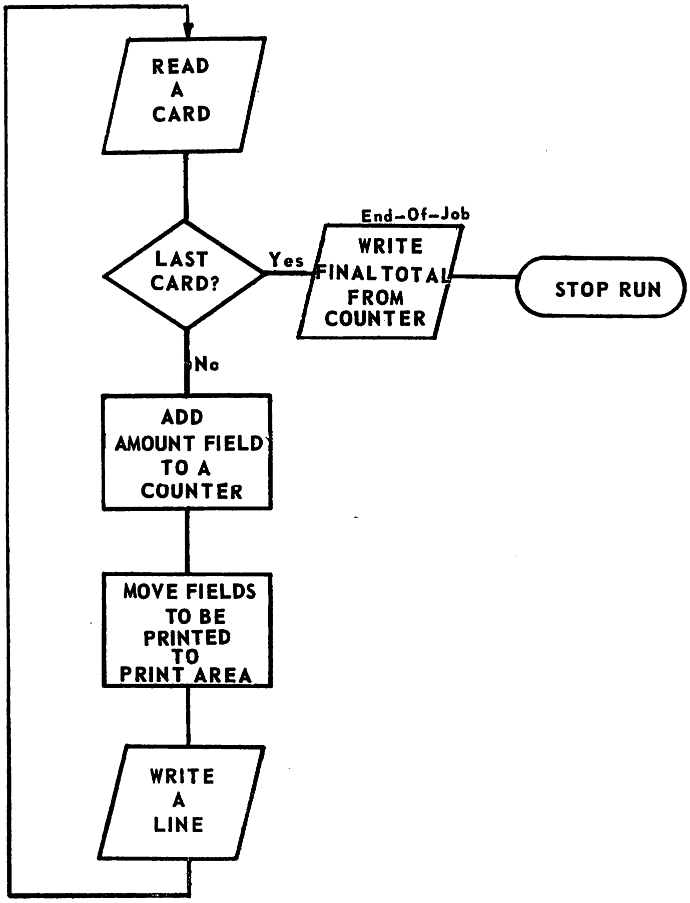

READ A CARD
LAST CARD?
Yes
End-Of-Job
WRITE FINAL TOTAL FROM COUNTER
STOP RUN
No
ADD AMOUNT FIELD TO A COUNTER
MOVE FIELDS TO BE PRINTED TO PRINT AREA
WRITE A LINE

Figure 5-19 Logic for the taking of a Final Total

When taking final totals, the amounts punched on the cards, the answers developed as a result of calculations, or numeric literals are added to a "counter" as each card is processed. At the "End of Job" Routine the contents of the counter or counters which contain the final totals are printed out. A Counter is nothing more than an area established in storage to accumulate totals. This area of storage is established in the COBOL Program in a special WORKING-STORAGE SECTION of the program.

SAMPLE PROBLEM

To further illustrate applications involving the taking of final totals and the related COBOL programming techniques, a problem is illustrated in this chapter in which two final totals are printed after all cards have been processed.

Input for the sample problem consists of punched cards which contain an Item Number field, an Asset field, a Cost field, a Salvage field, and a Life field. Depreciation is to be calculated by subtracting the salvage value of the item from the cost of the item and dividing the resulting answer by the estimated life of the asset. At the conclusion of the processing, a final total is to be printed of all of the depreciation values which have been calculated as well as the number of input cards which have been processed.

Input

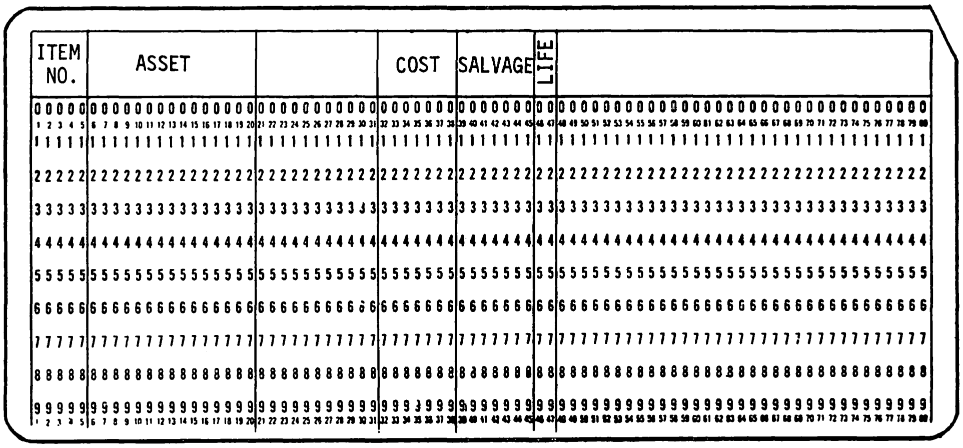
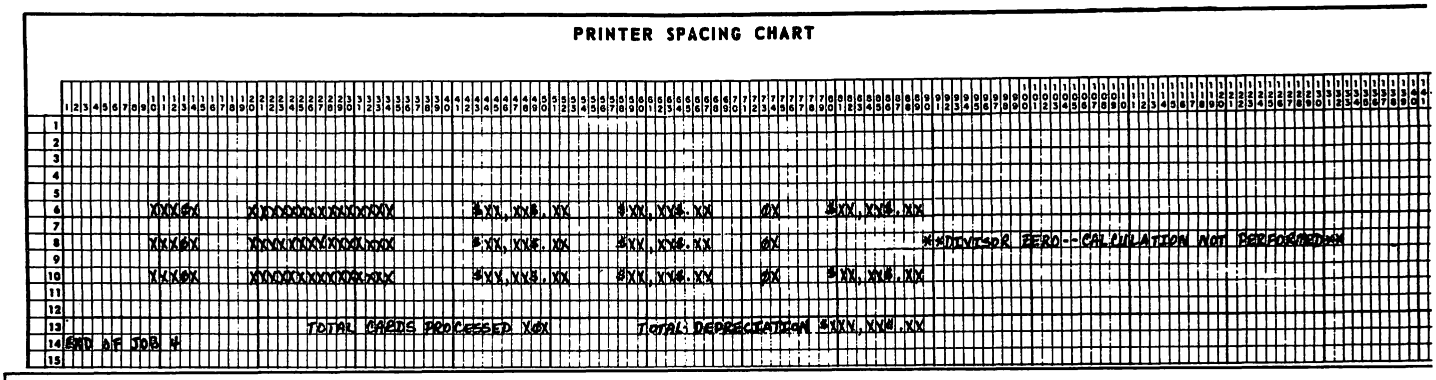


Figure 5-20 Input

Output

Output is to consist of a printed report. If the life of the asset is incorrectly punched as a zero, calculations cannot be performed correctly and an error message is printed on the report. In addition, a message is "displayed" on the console giving instructions to the computer operator. The printed report is illustrated in Figure 5-21.

Printed Report


PRINTER SPACING CHART
XXXØX
XX,XX.XX
ØX
DIVISOR ZERO--CALCULATION NOT PERFORMED
TOTAL CARDS PROCESSED XØX
TOTAL DEPRECIATION XXX,XX.XX
END OF JOB #

```
         10001     TRUCK                   $20,000.00     $1,000.00    10     $1,900.00

         10032     BUILDING                $57,952.30    $10,000.00    21     $2,283.44

         12546     DESKS                      $999.52        $53.20     6       $157.72

         12356     TYPEWRITER                 $150.00        $25.00     8        $15.63

         12356     CAR                        $999.99        $55.23     0              **DIVISOR ZERO--CALCULATION NOT PERFORMED**

         14561     CALCULATOR               $1,250.00       $100.00     9       $127.78

                         TOTAL CARDS PROCESSED    5      TOTAL DEPRECIATION   $4,484.57
END OF JOB 4
```

Figure 5-21 Printed Output Report

WORKING-STORAGE

The WORKING-STORAGE SECTION of the COBOL program may be used to reserve areas of storage for the "counters" which will be used to accumulate the "TOTAL CARDS PROCESSED" and the "TOTAL DEPRECIATION" in the sample problem. In addition, an area must be established in Working Storage to contain the answer resulting when Depreciation is calculated. This is necessary because the result of this calculation is to be added to a counter to accumulate a final total. Data that is to be added into a counter must be numeric, and cannot, therefore, contain any edit symbols such as the dollar sign, comma, or decimal point. Thus, in this problem calculations cannot be performed and the answer developed directly in the output line as in previous chapters.

The following diagram illustrates the basic steps when taking a final total.

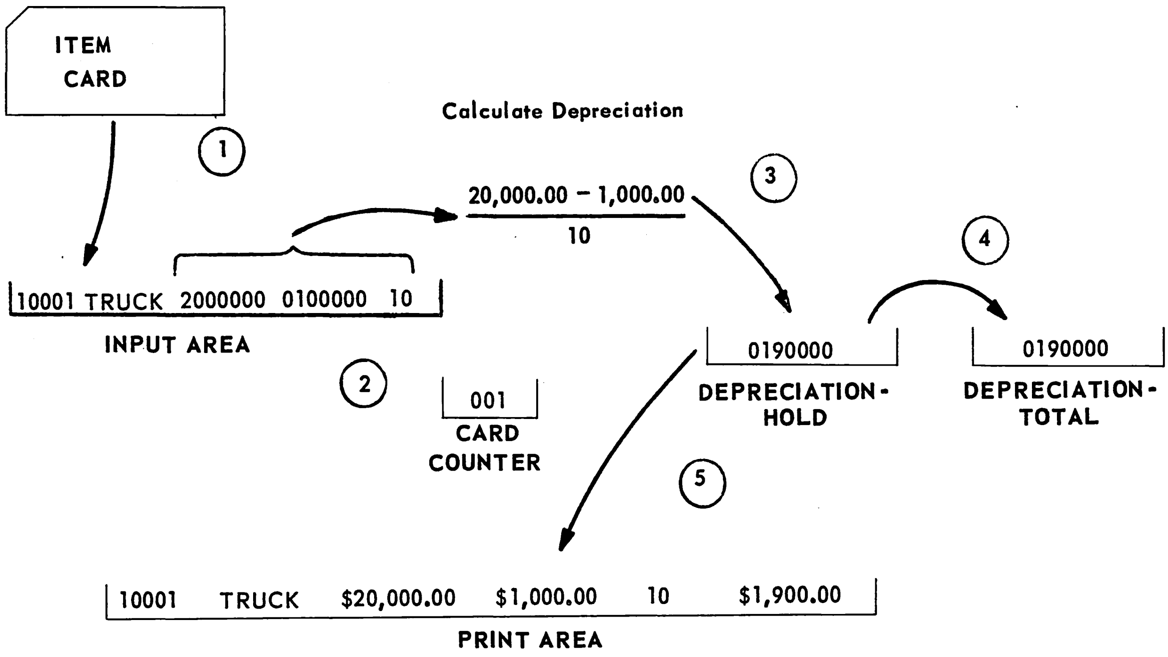

ITEM CARD
1
Calculate Depreciation
20,000.00 - 1,000.00
10
3
4
10001 TRUCK 2000000 0100000 10
INPUT AREA
2
001
CARD COUNTER
0190000
DEPRECIATION-HOLD
0190000
DEPRECIATION-TOTAL
5
10001 TRUCK $20,000.00 $1,000.00 10 $1,900.00
PRINT AREA

Figure 5-22 Steps in Taking a Final Total

1. A card is read into the Input Area.

2. A "1" is added to the CARD-COUNTER. This is an area established in the WORKING-STORAGE SECTION of the COBOL program. After all cards have been processed, CARD-COUNTER will contain the number of cards processed.

3. The Salvage value is subtracted from the Item value and the difference is divided by the Life of the asset to determine the depreciation. The answer is then stored in the DEPRECIATION-HOLD area which is established in the WORKING-STORAGE SECTION of the program.

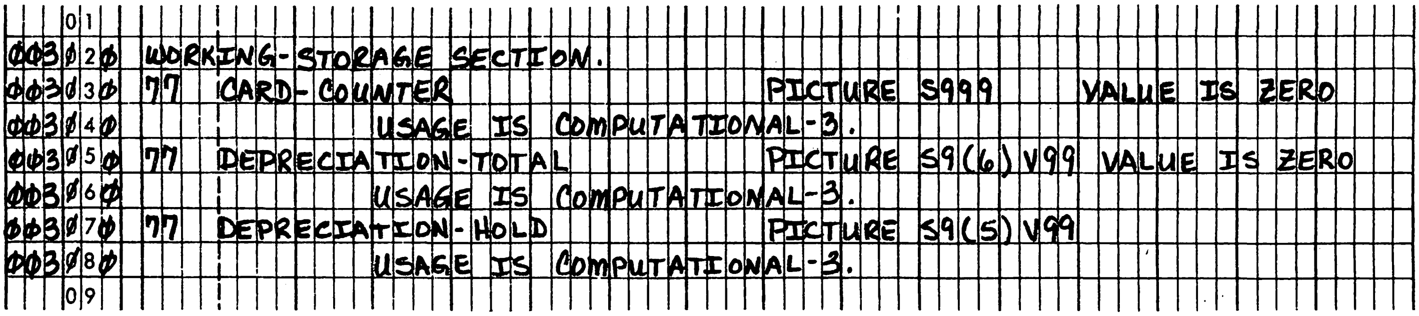
4. The contents of DEPRECIATION-HOLD are then added to the DEPRECIATION-TOTAL field which is also established in Working-Storage. After all of the cards have been processed, the DEPRECIATION-TOTAL field will contain the total of all of the depreciation values calculated.

5. Fields are moved to the Print Area, edited as required, and printed.

It should again be noted that fields that have been edited, that is, fields that contain a dollar sign, comma, decimal point, etc. cannot be added into counters set up in Working-Storage. Fields to be involved in calculations must be numeric in format.

The following segment of the COBOL program illustrates the entries in the WORKING-STORAGE SECTION of the program to save and accumulate totals.

EXAMPLE



```
003020 WORKING-STORAGE SECTION.
003030 77  CARD-COUNTER              PICTURE S999   VALUE IS ZERO
003040            USAGE IS COMPUTATIONAL-3.
003050 77  DEPRECIATION-TOTAL        PICTURE S9(6)V99 VALUE IS ZERO
003060            USAGE IS COMPUTATIONAL-3.
003070 77  DEPRECIATION-HOLD         PICTURE S9(5)V99
003080            USAGE IS COMPUTATIONAL-3.
```

Figure 5-23 WORKING-STORAGE SECTION Entries

The WORKING-STORAGE SECTION is used to describe areas of storage reserved for intermediate processing of data. This Section consists of a series of Record Description Entries, each of which describes an item in a work area.

An independent Working-Storage entry describes a single item that is not subdivided and is not itself a subdivision of some other item. Each of these types of items is defined in a separate Record Description entry, which begins with the special Level Number 77. All independent Working-Storage entries must precede any items having any of the level numbers 01 through 49.

Note in the sample program that Level Number 77 is used to specify each "counter". "DEPRECIATION-HOLD" is the name of the area which is to contain the depreciation after the calculations. "DEPRECIATION-TOTAL" is the name of the area that is to contain the final total of the depreciation amounts obtained as a result of the calculations, and "CARD-COUNTER" is the name of the area that is to be used to contain the final total of the number of cards processed.

Note the use of the S (sign), the VALUE Clause, and the USAGE IS Clause, in the PICTURES.

SIGN

The character "S" is used in the PICTURE Clause to indicate the presence of an operational sign. If used it must be the leftmost character of the Picture.

If a field contains a sign (such as an 11 zone punch in the units position of a numeric field to identify negative amounts), an "S" must be written as part of the PICTURE.

For answers developed by the execution of COBOL statements, the compiler will develop a negative sign if and only if an "S" is written in the picture. If an "S" is not written, answers developed will have a sign bit configuration interpreted as positive. Thus, in a subtraction operation if there is the possibility of an answer being negative, an "S" must be included in the PICTURE clause.

The programmer should include a sign in a PICTURE clause unless absolute values are desired. The use of unsigned fields increases the possibility of error if fields or answers developed could be negative. Also, additional code is generated to control the sign if a sign is not specified in the PICTURE clause. For numeric fields specified as unsigned (no S in the PICTURE clause) the COBOL compiler ensures that a sign is present so that the values are treated as absolute.

VALUE CLAUSE

The VALUE clause may be used to specify the initial value of WORKING-STORAGE items. In the sample problem "VALUE IS ZERO" is specified for each of the counters being used. "ZERO" or "ZEROES" is a Figurative Constant. This entry is necessary to insure that the contents of the area referenced by DEPRECIATION-TOTAL and CARD-COUNTER are set to zero at the beginning of the execution of the program. When an initial value is not specified, NO ASSUMPTION should be made regarding the initial contents of an item in WORKING-STORAGE.

The format of the VALUE clause is:

```
[VALUE IS literal]
```

Figure 5-24 Format Notation – VALUE Clause

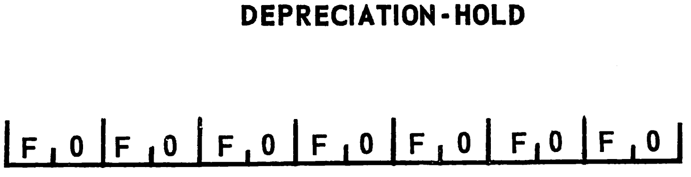
The size of the literal given in a VALUE Clause must be less than or equal to the size of the item as given in the PICTURE Clause with the provision that the literal must also include leading or trailing zeros to reflect P's in the PICTURE.

If the literal specified is a figurative constant, the size of the item generated is the size specified in the PICTURE Clause.

USAGE CLAUSE

The USAGE clause describes the form in which data is represented. In the sample problem the entries for the "counters" in WORKING-STORAGE include "USAGE IS COMPUTATIONAL-3". The clause will cause the areas referenced by DEPRECIATION-HOLD, DEPRECIATION-TOTAL, and CARD-COUNTER to be established in an Internal Decimal Format or COMPUTATIONAL-3 format.

If an area is reserved in WORKING-STORAGE without specifying COMPUTATIONAL-3, the area is set up in the External Decimal Format (DISPLAY). The following example illustrates how the area used for DEPRECIATION-HOLD would appear in storage if COMPUTATIONAL-3 is NOT specified in the PICTURE clause.



Note: The format shown is that used by computers using the Extended Binary Coded Decimal Interchange Code (EBCDIC).

Figure 5-25 Zeros Represented in DISPLAY Format

Note that 7 bytes of storage are reserved and each of the bytes contains a zero. Note that in this DISPLAY format, one position in storage is used for each digit in the field. When the field is converted to Computational-3, however, all of the sign positions of the bytes are dropped except for the low-order sign which is the sign for the field. This is illustrated below.

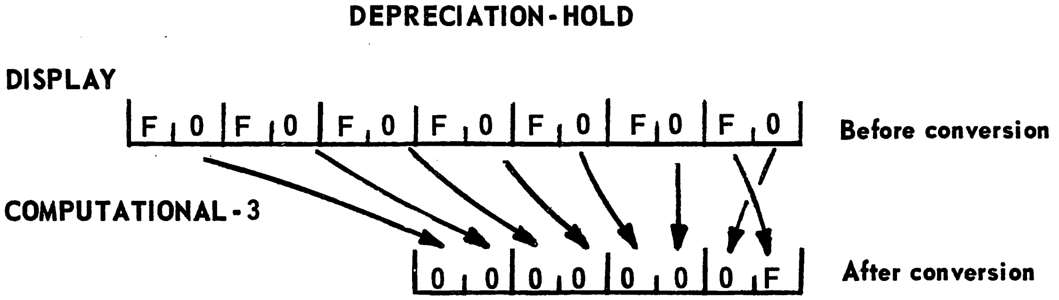


Figure 5-26 Converting Numeric Data from DISPLAY Format to COMPUTATIONAL-3 Format

If COMPUTATIONAL-3 is specified in the PICTURE clause, the area generated will be in an Internal Decimal format. It should be noted that the Computational-3 format is not found on all computers. It is found, however, on a number of machines used for business applications such as the IBM System/360, IBM System/370 and the NCR CENTURY series.

The format of the USAGE clause is illustrated below.

```
┌                                        ┐
                  INDEX
                  DISPLAY
                 {COMPUTATIONAL   }
  [USAGE IS]     {COMP            }
                 {COMPUTATIONAL-3 }
                 {COMP-3          }
└                                        ┘
```

Figure 5-27 Format-Notation – USAGE Clause

The INDEX option is used for special fields in table look-up operations and will be explained in a later chapter.

The DISPLAY option specifies that the item is stored in the External Decimal Format.

The COMPUTATIONAL or COMP option specifies that the item is stored in a binary format.

The COMPUTATIONAL-3 or COMP-3 option specifies that the item is stored in an internal decimal format as illustrated previously.

The USAGE clause may be written at any level. At a group level, it applies to each elementary item in the group. The usage of an elementary item must not contradict the usage explicitly stated for a group to which the item belongs. If USAGE is not specified, the usage of an item is assumed to be DISPLAY.

TOTAL LINES

The sample problem also illustrates the printing of a "total line" after all cards have been processed. The following diagram again illustrates the output to be obtained.

```
          10001     TRUCK                $20,000.00     $1,000.00    10     $1,900.00
          10032     BUILDING             $57,952.30    $10,000.00    21     $2,283.44
          12546     DESKS                   $999.52        $53.20     6       $157.72
          12356     TYPEWRITER              $150.00        $25.00     8        $15.63
          12356     CAR                     $999.99        $55.23     0              **DIVISOR ZERO--CALCULATION NOT PERFORMED**
          14561     CALCULATOR            $1,250.00       $100.00     9       $127.78

                         TOTAL CARDS PROCESSED   5        TOTAL DEPRECIATION  $4,484.57
END OF JOB 4
```

Figure 5-28 Printed Report

To print the "total line", the format of the line must be specified in the WORKING-STORAGE SECTION of the COBOL Program. The following diagram illustrates the complete WORKING-STORAGE SECTION of the sample program.

EXAMPLE

```
003020 WORKING-STORAGE SECTION.
003030 77  CARD-COUNTER                      PICTURE S999        VALUE IS ZERO
003040                 USAGE IS COMPUTATIONAL-3.
003050 77  DEPRECIATION-TOTAL                PICTURE S9(6)V99 VALUE IS ZERO
003060                 USAGE IS COMPUTATIONAL-3.
003070 77  DEPRECIATION-HOLD                 PICTURE S9(5)V99
003080                 USAGE IS COMPUTATIONAL-3.
003090 01  TOTAL-LINE.
003100     03  FILLER                        PICTURE X.
003110     03  FILLER                        PICTURE X(25)       VALUE SPACES.
003120     03  FILLER                        PICTURE X(22)
003130                 VALUE 'TOTAL CARDS PROCESSED '.
003140     03  CARD-TOT                      PICTURE ZZ9.
003150     03  FILLER                        PICTURE X(9)        VALUE SPACES.
003160     03  FILLER                        PICTURE X(19)
003170                 VALUE 'TOTAL DEPRECIATION '.
003180     03  DEPRECIATION-TOT              PICTURE $$$$,$$$.99.
003190     03  FILLER                        PICTURE X(43)       VALUE SPACES.
```

Figure 5-29 WORKING-STORAGE Entries to Describe TOTAL-LINE

Note that the line which is to be printed after all cards have been processed is specified by the Level 01 entry "01 TOTAL-LINE". This is followed by the Level 03 entries which define each portion of the TOTAL-LINE. The first entry, 03 FILLER with a PICTURE X, is required for the carriage control character, as with previous output lines. The next entry the 03 FILLER with PICTURE X(25) VALUE SPACES indicates that the first twenty-five positions on the paper are to be blank. See the Printer Spacing Chart in Figure 5-21. Note the use of VALUE SPACES. This entry is required to insure that the area does not contain any extraneous data from previous operations. Remember that no assumption should be made about the initial value of WORKING-STORAGE Items when these items have not had their initial values defined in a VALUE clause.

To print the identifying message "TOTAL CARDS PROCESSED" on the printed report a Level 03 entry with a FILLER, a PICTURE X(22) and the message TOTAL CARDS PROCESSED is specified.

The next line the "03 CARD-TOT" entry defines the area in which the contents of the CARD-COUNTER are to be moved and printed.

The entries on Lines 150 through Line 190 in Figure 5-29 describe the remaining portion of the "total line" in which the message "TOTAL DEPRECIATION" and the final total of the depreciation amount is printed.

WRITE – From Option

The WRITE statement CANNOT be used to directly reference data described in WORKING-STORAGE. That is, the statement "WRITE TOTAL-LINE" is invalid.

A WRITE statement must reference the output area established in the File Section of the Data Division in the COBOL program. Therefore, to print TOTAL-LINE, the area referenced by TOTAL-LINE must be moved to this output area. The following diagram illustrates this concept.

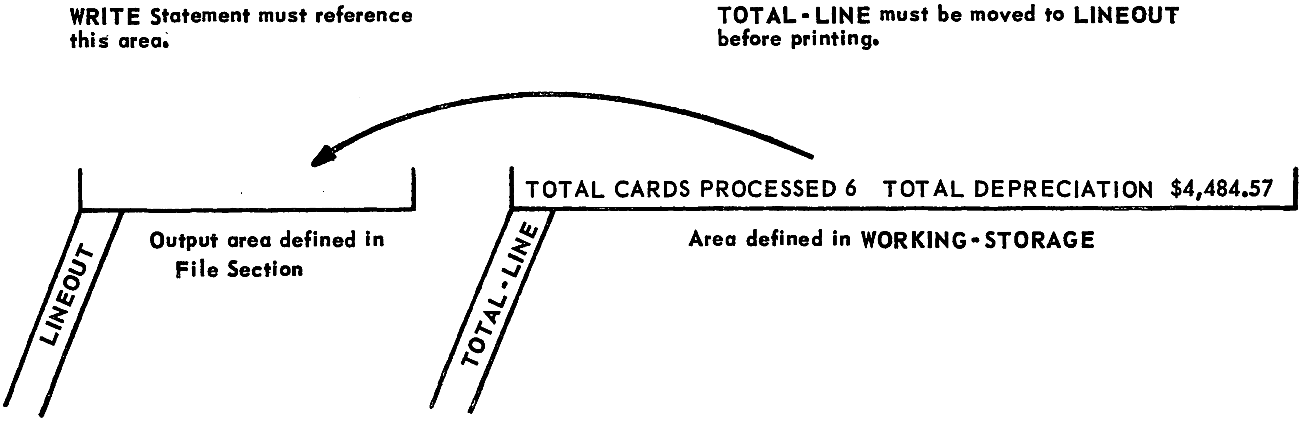

WRITE Statement must reference this area.
TOTAL-LINE must be moved to LINEOUT before printing.
TOTAL CARDS PROCESSED 6 TOTAL DEPRECIATION $4,484.57
Output area defined in File Section
Area defined in WORKING-STORAGE
LINEOUT
TOTAL-LINE

Figure 5-30 Diagram of Output Area and WORKING-STORAGE

The WRITE statement to print the contents of the area described by TOTAL-LINE is illustrated below.

```
WRITE LINEOUT FROM TOTAL-LINE AFTER ADVANCING 3 LINES.
CLOSE INVENTORY-CARDS, INVENTORY-CARD-LIST.
STOP RUN.
```

Figure 5-31 WRITE Statement – From Option

When the FROM Option is used in the WRITE statement, it is equivalent to both a MOVE statement and a WRITE statement. In the example illustrated above the identifier following the FROM is moved to the area referenced by the identifier specified before the FROM and printing takes places from the area specified by the first identifier. Thus, the effect of the WRITE statement illustrated above is to move TOTAL-LINE to LINEOUT, and print from the area referenced by LINEOUT.

STOP STATEMENT

The STOP statement is used to terminate or delay execution of the object program and has been used as the concluding statement in the previous examples. The format of the STOP statement is:

$$\underline{\texttt{STOP}}\ \left\{\begin{matrix}\underline{\texttt{RUN}}\\ \texttt{literal}\end{matrix}\right\}$$

Figure 5-32 STOP Statement

Use of the STOP LITERAL option will cause the literal specified to be displayed on the console. In addition, execution of the object program will pause. The program may be resumed by operator intervention or the execution of the program may be terminated at the option of the operator.

The following is an example of the STOP statement and the resulting output on the console typewriter.

```
004130     STOP 'CONSULT RUN BOOK---JOB 4'.
```

```
ERROR IN PROCESSING JOB 4, DEPRECIATION NOT CALCULATED
CONSULT RUN BOOK---JOB 4
```

Figure 5-33 Example of STOP Statement

SAMPLE PROBLEM

The following pages illustrate the complete Input, Output, Flowchart, and COBOL program for the calculation of the depreciation and also the taking of final totals.

Input

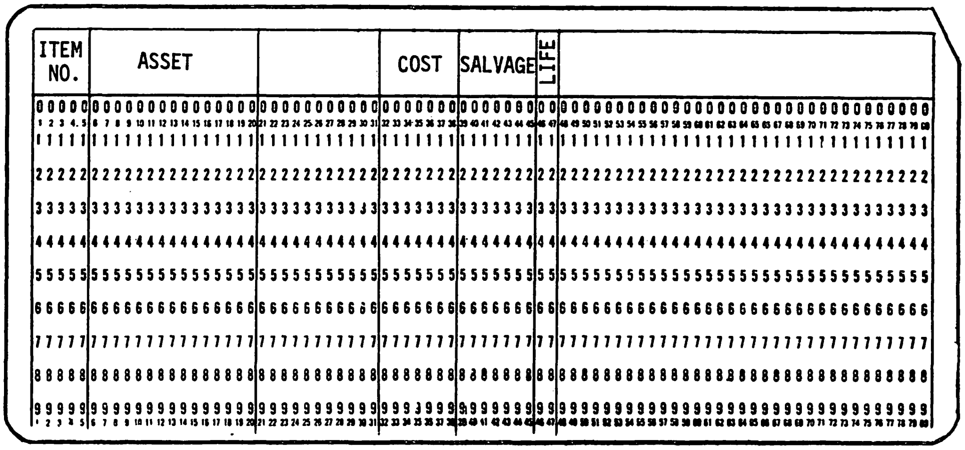
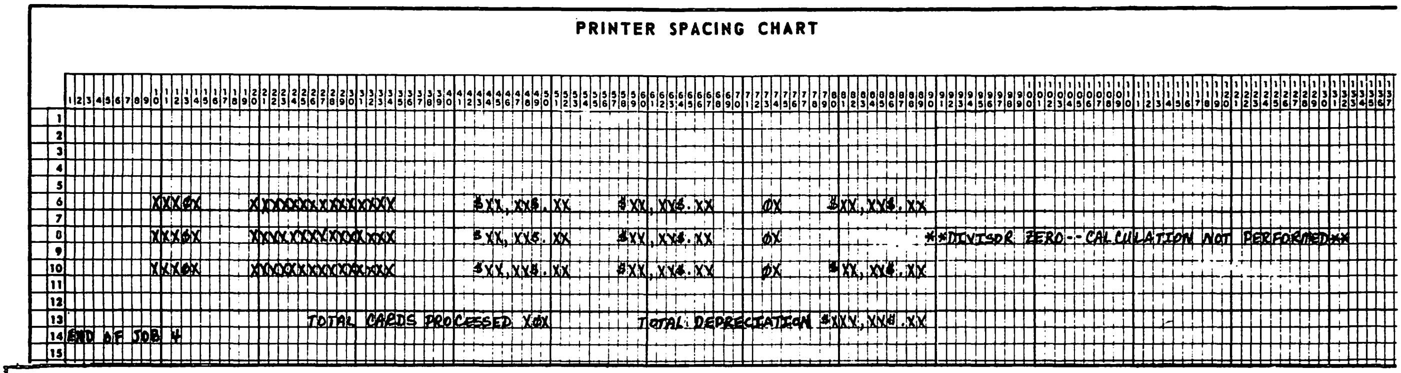

ITEM NO.
ASSET
COST
SALVAGE
LIFE

Figure 5-34 Input

Output


PRINTER SPACING CHART
XXXØX
XXXXXXXXXXXXXXX
XX,XX.XX
ØX
DIVISOR ZERO--CALCULATION NOT PERFORMED
TOTAL CARDS PROCESSED XØX
TOTAL DEPRECIATION XXX,XX.XX
END OF JOB 4

```
        10001    TRUCK              $20,000.00     $1,000.00    10    $1,900.00
        10032    BUILDING           $57,952.30    $10,000.00    21    $2,283.44
        12546    DESKS                 $999.52        $53.20     6      $157.72
        12356    TYPEWRITER            $150.00        $25.00     8       $15.63
        12356    CAR                   $999.99        $55.23     0                 **DIVISOR ZERO--CALCULATION NOT PERFORMED**
        14561    CALCULATOR          $1,250.00       $100.00     9      $127.78

                      TOTAL CARDS PROCESSED   5         TOTAL DEPRECIATION   $4,484.57
END OF JOB 4
```

Figure 5-35 Output

FLOWCHART

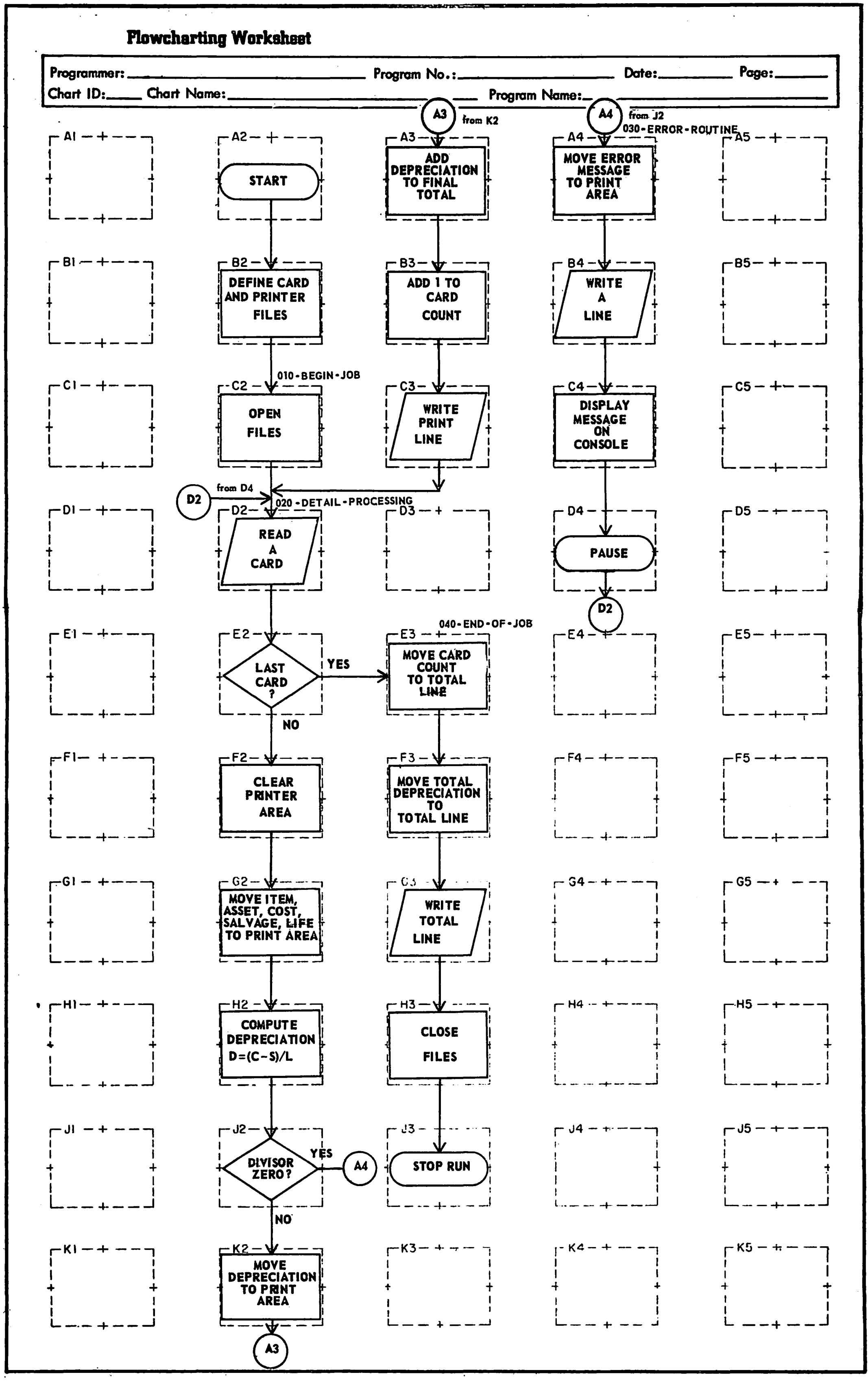
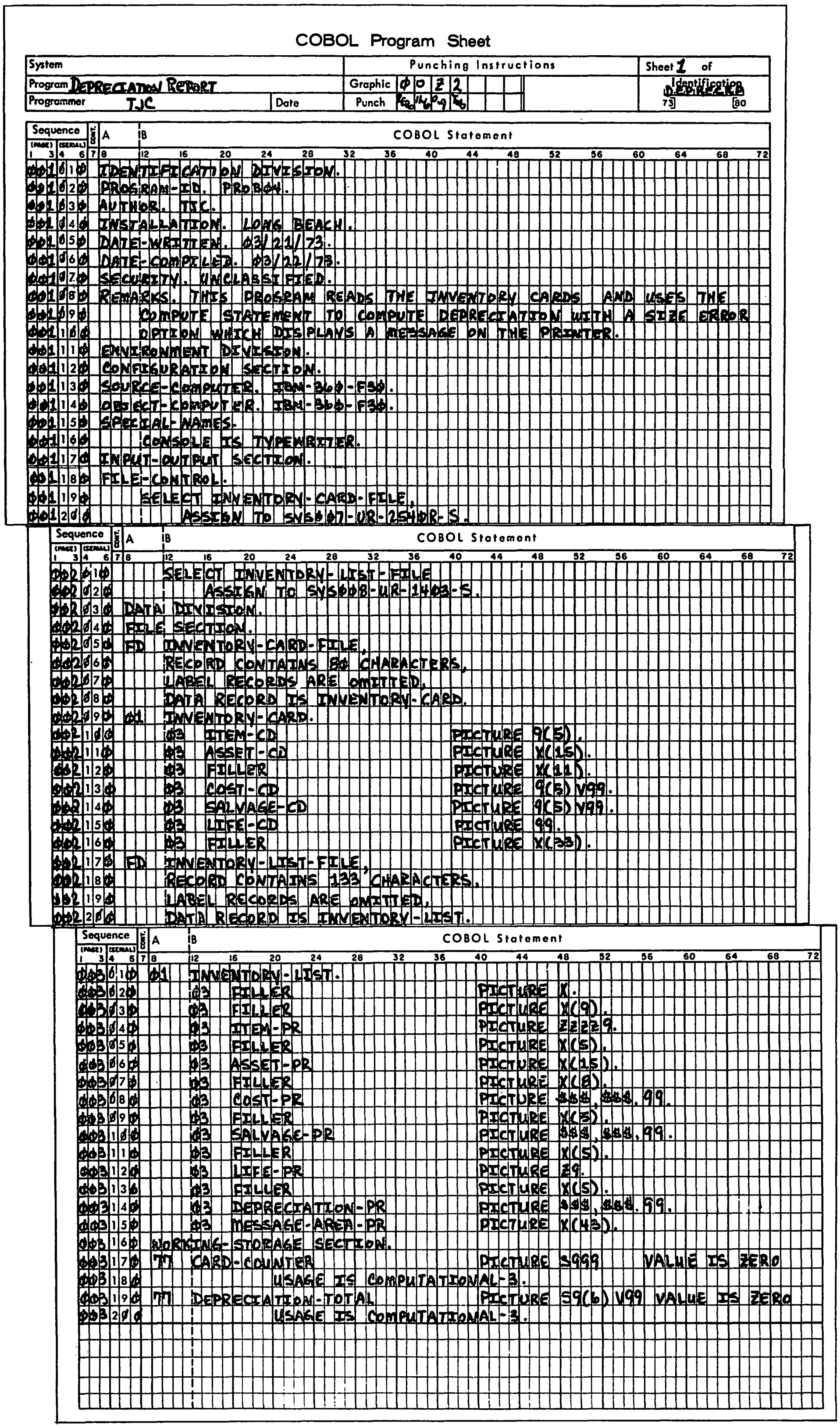

Flowcharting Worksheet
Programmer:
Program No.:
Date:
Page:
Chart ID:
Chart Name:
Program Name:
A1 A2 A3 A4 A5
B1 B2 B3 B4 B5
C1 C2 C3 C4 C5
D1 D2 D3 D4 D5
E1 E2 E3 E4 E5
F1 F2 F3 F4 F5
G1 G2 G3 G4 G5
H1 H2 H3 H4 H5
J1 J2 J3 J4 J5
K1 K2 K3 K4 K5
A3
from K2
A4
from J2
030-ERROR-ROUTINE
START
DEFINE CARD AND PRINTER FILES
010-BEGIN-JOB
OPEN FILES
D2
from D4
020-DETAIL-PROCESSING
READ A CARD
LAST CARD ?
YES
NO
CLEAR PRINTER AREA
MOVE ITEM, ASSET, COST, SALVAGE, LIFE TO PRINT AREA
COMPUTE DEPRECIATION D=(C-S)/L
DIVISOR ZERO?
YES
A4
NO
MOVE DEPRECIATION TO PRINT AREA
A3
ADD DEPRECIATION TO FINAL TOTAL
ADD 1 TO CARD COUNT
WRITE PRINT LINE
040-END-OF-JOB
MOVE CARD COUNT TO TOTAL LINE
MOVE TOTAL DEPRECIATION TO TOTAL LINE
WRITE TOTAL LINE
CLOSE FILES
STOP RUN
MOVE ERROR MESSAGE TO PRINT AREA
WRITE A LINE
DISPLAY MESSAGE ON CONSOLE
PAUSE
D2

Figure 5-36 Program Flowchart

SOURCE CODING – SAMPLE PROGRAM



COBOL Program Sheet

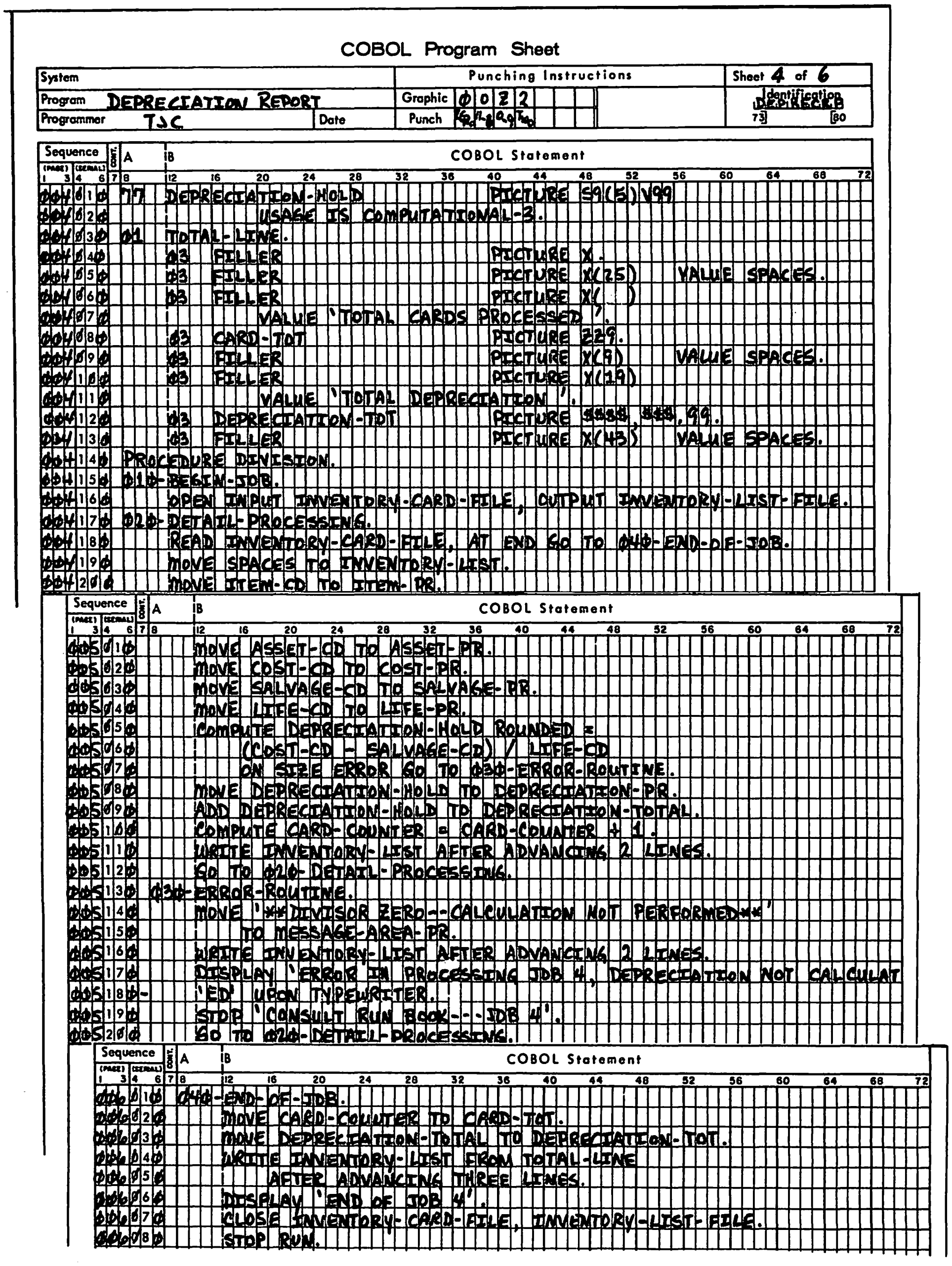
System	Punching Instructions	Sheet 1 of
Program DEPRECIATION REPORT	Graphic 0 O Z 2	Identification DEPRECRB
Programmer TJC Date	Punch [illegible]	73 80

```
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. PROB04.
001030 AUTHOR. TJC.
001040 INSTALLATION. LONG BEACH.
001050 DATE-WRITTEN. 03/21/73.
001060 DATE-COMPILED. 03/22/73.
001070 SECURITY. UNCLASSIFIED.
001080 REMARKS. THIS PROGRAM READS THE INVENTORY CARDS  AND USES THE
001090     COMPUTE STATEMENT TO COMPUTE DEPRECIATION WITH A SIZE ERROR
001100     OPTION WHICH DISPLAYS A MESSAGE ON THE PRINTER.
001110 ENVIRONMENT DIVISION.
001120 CONFIGURATION SECTION.
001130 SOURCE-COMPUTER. IBM-360-F30.
001140 OBJECT-COMPUTER. IBM-360-F30.
001150 SPECIAL-NAMES.
001160     CONSOLE IS TYPEWRITER.
001170 INPUT-OUTPUT SECTION.
001180 FILE-CONTROL.
001190     SELECT INVENTORY-CARD-FILE,
001200         ASSIGN TO SYS007-UR-2540R-S.
```

```
002010     SELECT INVENTORY-LIST-FILE
002020         ASSIGN TO SYS008-UR-1403-S.
002030 DATA DIVISION.
002040 FILE SECTION.
002050 FD  INVENTORY-CARD-FILE,
002060     RECORD CONTAINS 80 CHARACTERS,
002070     LABEL RECORDS ARE OMITTED,
002080     DATA RECORD IS INVENTORY-CARD.
002090 01  INVENTORY-CARD.
002100     03  ITEM-CD                 PICTURE 9(5).
002110     03  ASSET-CD                PICTURE X(15).
002120     03  FILLER                  PICTURE X(11).
002130     03  COST-CD                 PICTURE 9(5)V99.
002140     03  SALVAGE-CD              PICTURE 9(5)V99.
002150     03  LIFE-CD                 PICTURE 99.
002160     03  FILLER                  PICTURE X(33).
002170 FD  INVENTORY-LIST-FILE,
002180     RECORD CONTAINS 133 CHARACTERS,
002190     LABEL RECORDS ARE OMITTED,
002200     DATA RECORD IS INVENTORY-LIST.
```

```
003010 01  INVENTORY-LIST.
003020     03  FILLER                  PICTURE X.
003030     03  FILLER                  PICTURE X(9).
003040     03  ITEM-PR                 PICTURE ZZZZ9.
003050     03  FILLER                  PICTURE X(5).
003060     03  ASSET-PR                PICTURE X(15).
003070     03  FILLER                  PICTURE X(8).
003080     03  COST-PR                 PICTURE $$$,$$$.99.
003090     03  FILLER                  PICTURE X(5).
003100     03  SALVAGE-PR              PICTURE $$$,$$$.99.
003110     03  FILLER                  PICTURE X(5).
003120     03  LIFE-PR                 PICTURE Z9.
003130     03  FILLER                  PICTURE X(5).
003140     03  DEPRECIATION-PR         PICTURE $$$,$$$.99.
003150     03  MESSAGE-AREA-PR         PICTURE X(43).
003160 WORKING-STORAGE SECTION.
003170 77  CARD-COUNTER                PICTURE S999     VALUE IS ZERO
003180             USAGE IS COMPUTATIONAL-3.
003190 77  DEPRECIATION-TOTAL          PICTURE S9(6)V99 VALUE IS ZERO
003200             USAGE IS COMPUTATIONAL-3.
```

Figure 5-37 Source Coding – Part 1 of 2



COBOL Program Sheet

System
Program DEPRECIATION REPORT
Programmer TJC
Date
Punching Instructions
Graphic 0 0 Z 2
Punch
Sheet 4 of 6
Identification DEPRECRB 73 80

```
004010 77  DEPRECIATION-HOLD           PICTURE S9(5)V99
004020             USAGE IS COMPUTATIONAL-3.
004030 01  TOTAL-LINE.
004040     03  FILLER                  PICTURE X.
004050     03  FILLER                  PICTURE X(25)   VALUE SPACES.
004060     03  FILLER                  PICTURE X(  )
004070             VALUE 'TOTAL CARDS PROCESSED '.
004080     03  CARD-TOT                PICTURE ZZ9.
004090     03  FILLER                  PICTURE X(9)    VALUE SPACES.
004100     03  FILLER                  PICTURE X(19)
004110             VALUE 'TOTAL DEPRECIATION '.
004120     03  DEPRECIATION-TOT        PICTURE $$$$,$$$.99.
004130     03  FILLER                  PICTURE X(43)   VALUE SPACES.
004140 PROCEDURE DIVISION.
004150 010-BEGIN-JOB.
004160     OPEN INPUT INVENTORY-CARD-FILE, OUTPUT INVENTORY-LIST-FILE.
004170 020-DETAIL-PROCESSING.
004180     READ INVENTORY-CARD-FILE, AT END GO TO 040-END-OF-JOB.
004190     MOVE SPACES TO INVENTORY-LIST.
004200     MOVE ITEM-CD TO ITEM-PR.
005010     MOVE ASSET-CD TO ASSET-PR.
005020     MOVE COST-CD TO COST-PR.
005030     MOVE SALVAGE-CD TO SALVAGE-PR.
005040     MOVE LIFE-CD TO LIFE-PR.
005050     COMPUTE DEPRECIATION-HOLD ROUNDED =
005060         (COST-CD - SALVAGE-CD) / LIFE-CD
005070         ON SIZE ERROR GO TO 030-ERROR-ROUTINE.
005080     MOVE DEPRECIATION-HOLD TO DEPRECIATION-PR.
005090     ADD DEPRECIATION-HOLD TO DEPRECIATION-TOTAL.
005100     COMPUTE CARD-COUNTER = CARD-COUNTER + 1.
005110     WRITE INVENTORY-LIST AFTER ADVANCING 2 LINES.
005120     GO TO 020-DETAIL-PROCESSING.
005130 030-ERROR-ROUTINE.
005140     MOVE '**DIVISOR ZERO--CALCULATION NOT PERFORMED**'
005150         TO MESSAGE-AREA-PR.
005160     WRITE INVENTORY-LIST AFTER ADVANCING 2 LINES.
005170     DISPLAY 'ERROR IN PROCESSING JOB 4, DEPRECIATION NOT CALCULAT
005180-    'ED' UPON TYPEWRITER.
005190     STOP 'CONSULT RUN BOOK---JOB 4'.
005200     GO TO 020-DETAIL-PROCESSING.
006010 040-END-OF-JOB.
006020     MOVE CARD-COUNTER TO CARD-TOT.
006030     MOVE DEPRECIATION-TOTAL TO DEPRECIATION-TOT.
006040     WRITE INVENTORY-LIST FROM TOTAL-LINE
006050         AFTER ADVANCING THREE LINES.
006060     DISPLAY 'END OF JOB 4'.
006070     CLOSE INVENTORY-CARD-FILE, INVENTORY-LIST-FILE.
006080     STOP RUN.
```

Figure 5-38 Source Coding – Part 2 of 2

```
        1                          IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4              03/22/73

00001   001010 IDENTIFICATION DIVISION.                                         DEPRECRP
00002   001020 PROGRAM-ID. PROB04.                                              DEPRECRP
00003   001030 AUTHOR. TJC.                                                     DEPRECRP
00004   001040 INSTALLATION. LONG BEACH.                                        DEPRECRP
00005   001050 DATE-WRITTEN. 03/21/73.                                          DEPRECRP
00006   001060 DATE-COMPILED. 03/22/73.                                         DEPRECRP
00007   001070 SECURITY. UNCLASSIFIED.                                          DEPRECRP
00008   001080 REMARKS. THIS PROGRAM READS THE INVENTORY CARDS  AND USES THE    DEPRECRP
00009   001090     COMPUTE STATEMENT TO COMPUTE DEPRECIATION WITH A SIZE ERROR  DEPRECRP
00010   001100     OPTION WHICH DISPLAYS A MESSAGE ON THE PRINTER.              DEPRECRP
00011   001110 ENVIRONMENT DIVISION.                                            DEPRECRP
00012   001120 CONFIGURATION SECTION.                                           DEPRECRP
00013   001130 SOURCE-COMPUTER. IBM-360-F30.                                    DEPRECRP
00014   001140 OBJECT-COMPUTER. IBM-360-F30.                                    DEPRECRP
00015   001150 SPECIAL-NAMES.                                                   DEPRECRP
00016   001160     CONSOLE IS TYPEWRITER.                                       DEPRECRP
00017   001170 INPUT-OUTPUT SECTION.                                            DEPRECRP
00018   001180 FILE-CONTROL.                                                    DEPRECRP
00019   001190     SELECT INVENTORY-CARD-FILE                                   DEPRECRP
00020   001200         ASSIGN TO SYS007-UR-2540R-S.                             DEPRECRP
00021   002010     SELECT INVENTORY-LIST-FILE                                   DEPRECRP
00022   002020         ASSIGN TO SYS008-UR-1403-S.                              DEPRECRP
00023   002030 DATA DIVISION.                                                   DEPRECRP
00024   002040 FILE SECTION.                                                    DEPRECRP
00025   002050 FD  INVENTORY-CARD-FILE,                                         DEPRECRP
00026   002060     RECORD CONTAINS 80 CHARACTERS,                               DEPRECRP
00027   002070     LABEL RECORDS ARE OMITTED,                                   DEPRECRP
00028   002080     DATA RECORD IS INVENTORY-CARD.                               DEPRECRP
00029   002090 01  INVENTORY-CARD.                                              DEPRECRP
00030   002100     03  ITEM-CD                 PICTURE 9(5).                    DEPRECRP
00031   002110     03  ASSET-CD                PICTURE X(15).                   DEPRECRP
00032   002120     03  FILLER                  PICTURE X(11).                   DEPRECRP
00033   002130     03  COST-CD                 PICTURE 9(5)V99.                 DEPRECRP
00034   002140     03  SALVAGE-CD              PICTURE 9(5)V99.                 DEPRECRP
00035   002150     03  LIFE-CD                 PICTURE 99.                      DEPRECRP
00036   002160     03  FILLER                  PICTURE X(33).                   DEPRECRP
00037   002170 FD  INVENTORY-LIST-FILE,                                         DEPRECRP
00038   002180     RECORD CONTAINS 133 CHARACTERS,                              DEPRECRP
00039   002190     LABEL RECORDS ARE OMITTED,                                   DEPRECRP
00040   002200     DATA RECORD IS INVENTORY-LIST.                               DEPRECRP
00041   003010 01  INVENTORY-LIST.                                              DEPRECRP
00042   003020     03  FILLER                  PICTURE X.                       DEPRECRP
00043   003030     03  FILLER                  PICTURE X(9).                    DEPRECRP
00044   003040     03  ITEM-PR                 PICTURE ZZZZ9.                   DEPRECRP
00045   003050     03  FILLER                  PICTURE X(5).                    DEPRECRP
00046   003060     03  ASSET-PR                PICTURE X(15).                   DEPRECRP
00047   003070     03  FILLER                  PICTURE X(8).                    DEPRECRP
00048   003080     03  COST-PR                 PICTURE $$$,$$$.99.              DEPRECRP
00049   003090     03  FILLER                  PICTURE X(5).                    DEPRECRP
00050   003100     03  SALVAGE-PR              PICTURE $$$,$$$.99.              DEPRECRP
00051   003110     03  FILLER                  PICTURE X(5).                    DEPRECRP
00052   003120     03  LIFE-PR                 PICTURE Z9.                      DEPRECRP
00053   003130     03  FILLER                  PICTURE X(5).                    DEPRECRP
00054   003140     03  DEPRECIATION-PR         PICTURE $$$,$$$.99.              DEPRECRP
00055   003150     03  MESSAGE-AREA-PR         PICTURE X(43).                   DEPRECRP
00056   003160 WORKING-STORAGE SECTION.                                         DEPRECRP
00057   003170 77  CARD-COUNTER                PICTURE S999     VALUE IS ZERO   DEPRECRP
00058   003180             USAGE IS COMPUTATIONAL-3.                            DEPRECRP
00059   003190 77  DEPRECIATION-TOTAL          PICTURE S9(6)V99 VALUE IS ZERO   DEPRECRP
00060   003200             USAGE IS COMPUTATIONAL-3.                            DEPRECRP
00061   004010 77  DEPRECIATION-HOLD           PICTURE S9(5)V99                 DEPRECRP
00062   004020             USAGE IS COMPUTATIONAL-3.                            DEPRECRP
00063   004030 01  TOTAL-LINE.                                                  DEPRECRP
00064   004040     03  FILLER                  PICTURE X.                       DEPRECRP
00065   004050     03  FILLER                  PICTURE X(25)    VALUE SPACES.   DEPRECRP
00066   004060     03  FILLER                  PICTURE X(22)                    DEPRECRP
00067   004070             VALUE 'TOTAL CARDS PROCESSED '.                      DEPRECRP
00068   004080     03 CARD-TOT                 PICTURE ZZ9.                     DEPRECRP
00069   004090     03  FILLER                  PICTURE X(9)     VALUE SPACES.   DEPRECRP
00070   004100     03  FILLER                  PICTURE X(19)                    DEPRECRP
00071   004110             VALUE 'TOTAL DEPRECIATION '.                         DEPRECRP
00072   004120     03  DEPRECIATION-TOT        PICTURE $$$$,$$$.99.             DEPRECRP
00073   004130     03  FILLER                  PICTURE X(43)    VALUE SPACES.   DEPRECRP
00074   004140 PROCEDURE DIVISION.                                              DEPRECRP
00075   004150 010-BEGIN-JOB.                                                   DEPRECRP
00076   004160     OPEN INPUT INVENTORY-CARD-FILE, OUTPUT INVENTORY-LIST-FILE.  DEPRECRP
00077   004170 020-DETAIL-PROCESSING.                                           DEPRECRP
00078   004180     READ INVENTORY-CARD-FILE, AT END GO TO 040-END-OF-JOB.       DEPRECRP
00079   004190     MOVE SPACES TO INVENTORY-LIST.                               DEPRECRP
00080   004200     MOVE ITEM-CD TO ITEM-PR.                                     DEPRECRP
00081   005010     MOVE ASSET-CD TO ASSET-PR.                                   DEPRECRP
00082   005020     MOVE COST-CD TO COST-PR.                                     DEPRECRP
00083   005030     MOVE SALVAGE-CD TO SALVAGE-PR.                               DEPRECRP
00084   005040     MOVE LIFE-CD TO LIFE-PR.                                     DEPRECRP
00085   005050     COMPUTE DEPRECIATION-HOLD ROUNDED =                          DEPRECRP
00086   005060         (COST-CD - SALVAGE-CD) / LIFE-CD                         DEPRECRP
00087   005070         ON SIZE ERROR GO TO 030-ERROR-ROUTINE.                   DEPRECRP
00088   005080     MOVE DEPRECIATION-HOLD TO DEPRECIATION-PR.                   DEPRECRP
00089   005090     ADD DEPRECIATION-HOLD TO DEPRECIATION-TOTAL.                 DEPRECRP
00090   005100     COMPUTE CARD-COUNTER = CARD-COUNTER + 1.                     DEPRECRP
00091   005110     WRITE INVENTORY-LIST AFTER ADVANCING 2 LINES.                DEPRECRP
00092   005120     GO TO 020-DETAIL-PROCESSING.                                 DEPRECRP
00093   005130 030-ERROR-ROUTINE.                                               DEPRECRP
00094   005140     MOVE '**DIVISOR ZERO--CALCULATION NOT PERFORMED**'           DEPRECRP
00095   005150         TO MESSAGE-AREA-PR.                                      DEPRECRP
00096   005160     WRITE INVENTORY-LIST AFTER ADVANCING 2 LINES.                DEPRECRP
00097   005170     DISPLAY 'ERROR IN PROCESSING JOB 4, DEPRECIATION NOT CALCULATDEPRECRP
00098   005180-    'ED'  UPON TYPEWRITER.                                       DEPRECRP
00099   005190     STOP 'CONSULT RUN BOOK---JOB 4'.                             DEPRECRP
00100   005200     GO TO 020-DETAIL-PROCESSING.                                 DEPRECRP
00101   006010 040-END-OF-JOB.                                                  DEPRECRP
00102   006020     MOVE CARD-COUNTER TO CARD-TOT.                               DEPRECRP
00103   006030     MOVE DEPRECIATION-TOTAL TO DEPRECIATION-TOT.                 DEPRECRP
00104   006040     WRITE INVENTORY-LIST FROM TOTAL-LINE                         DEPRECRP
00105   006050         AFTER ADVANCING 3 LINES.                                 DEPRECRP
00106   006060     DISPLAY 'END OF JOB 4'.                                      DEPRECRP
00107   006070     CLOSE INVENTORY-CARD-FILE, INVENTORY-LIST-FILE.              DEPRECRP
00108   006080     STOP RUN.                                                    DEPRECRP
```

Figure 5-39 Source Listing

CHAPTER 5

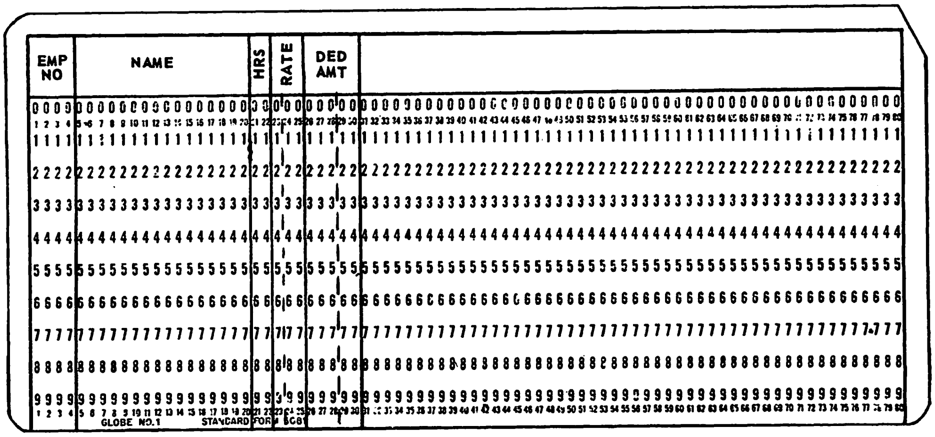
PROGRAMMING ASSIGNMENT 1

INSTRUCTIONS

Write the COBOL program to produce a Weekly Payroll Report.

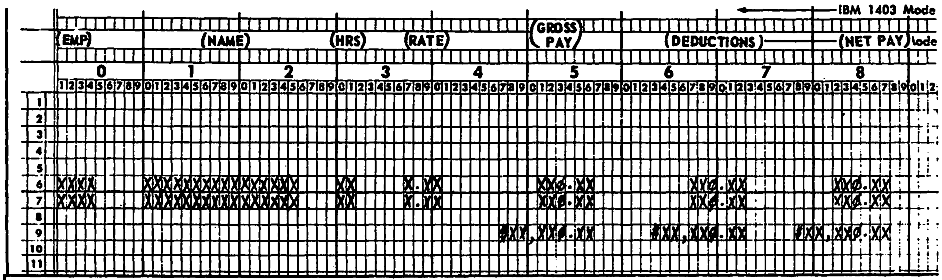
INPUT: Employee Payroll Cards

Input is to consist of Employee Payroll cards containing the Employee Number, Employee Name, Hours Worked, Rate of Pay, and Deductions. The format of the cards is illustrated below.


EMP NO
NAME
HRS
RATE
DED AMT
GLOBE NO.1
STANDARD FORM 5081

OUTPUT: Net Pay Report

Output is to consist of a Net Pay report listing the Employee Number, the Employee Name, the Hours Worked, Rate of Pay, Gross Pay, Deductions, and Net Pay. Gross Pay is to be calculated by multiplying the Rate of Pay by the Hours Worked. Net Pay is to be calculated by subtracting the Deductions from the calculated Gross Pay to produce the Net Pay. Final Totals are to be taken of Gross Pay, Deductions, and Net Pay.


IBM 1403 Mode
(EMP)
(NAME)
(HRS)
(RATE)
(GROSS PAY)
(DEDUCTIONS)
(NET PAY)
XXXX
XXXXXXXXXXXXXXXX
XX
X.XX
XX0.XX
$XX,XX0.XX

```
0105      BAKNKE, IGOR            40     2.00          80.00          15.00          65.00
1970      DUGGLE, TED             40     3.00         120.00          20.00         100.00
2658      JOHNSTON, JOHN M.       40     2.50         100.00          18.00          82.00
                                                          :              :              :
```

NOTE: Final Totals of Gross Pay, Deductions, Net Pay → $25,500.00 $5,000.00 $20,500.00

TEST DATA

EMP NO (col 1-4)	NAME (col 5-20)	HRS (col 21-22)	RATE (col 23-25)	DED AMT (col 26-30)
0105	BAKNKE, IGOR	40	200	01500
1970	DUGGLE, TED	40	300	02000
2658	JOHNSTON, JOHN M.	40	250	01880
3723	KILNER, ROBERT	35	375	02800
4922	McMAY, OSCAR	32	425	04200
5013	NELSEN, OLLIE	40	500	05000
6227	SAMSON, THOMAS	37	623	06150

CHAPTER 5

PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

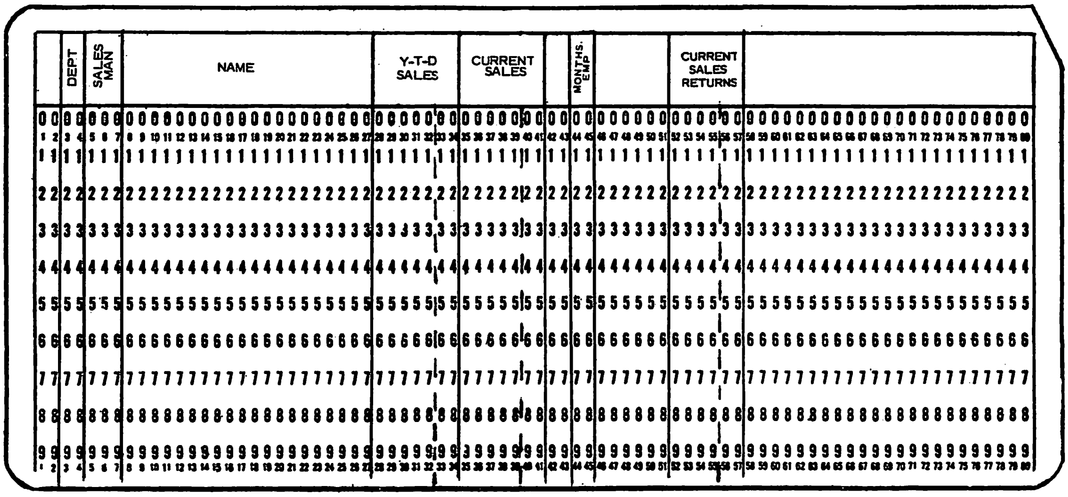
From the Sales Cards illustrated below a Monthly Sales Report is to be prepared. Average Monthly Sales are to be calculated by adding Y-T-D Sales and Current Sales, subtracting Current Sales Returns, and then dividing by Months Employed.

If Months Employed is zero or if Average Monthly Sales is greater than $9,999.99, print an error message on the report.

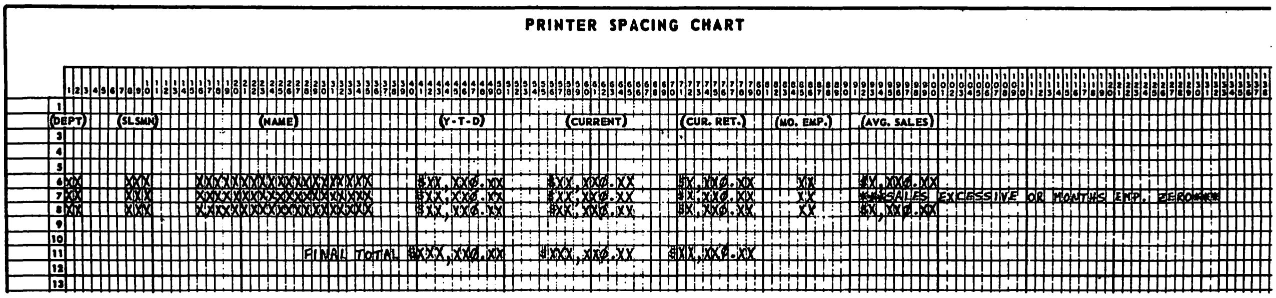
Cards are to be sorted by Salesman Number.

A final total is to be taken from the Y-T-D Sales, the Current Sales, and the Sales Returns. The test data is contained in the appendices.

INPUT: Sales Cards


DEPT
SALES MAN
NAME
Y-T-D SALES
CURRENT SALES
MONTHS EMP
CURRENT SALES RETURNS

OUTPUT: Monthly Sales Report


PRINTER SPACING CHART
(DEPT)
(SLSMN)
(NAME)
(Y-T-D)
(CURRENT)
(CUR. RET.)
(MO. EMP.)
(AVG. SALES)
XX XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX $XX,XX0.XX $X,XX0.XX XX $X,XX0.XX
XX XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX $XX,XX0.XX $X,XX0.XX XX **SALES EXCESSIVE OR MONTHS EMP. ZERO**
XX XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX $XX,XX0.XX $X,XX0.XX XX $X,XX0.XX
FINAL TOTAL $XXX,XX0.XX $XXX,XX0.XX $XX,XX0.XX

CHAPTER 6

COMPARING

INTRODUCTION

The ability of the computer to compare numeric values or letters of the alphabet and perform alternative operations based upon the results of this comparison provides electronic data processing systems with great flexibility in the solution of business and mathematical problems.

The following flowchart illustrates the logic necessary to solve a "typical" business type programming problem in which it is desired to compare the Current Sales Field punched on a card to the Sales Quota Field also punched on the same card.

The input cards contain the Employee Number, the Employee Name, the Current Sales, and the Sales Quota. If the Current Sales is greater than the Sales Quota a line is to be printed. If the Current Sales is less than the Sales Quota a line is to be printed with the identifying message "UNDER QUOTA". In addition, a card is to be punched containing those employees under quota.

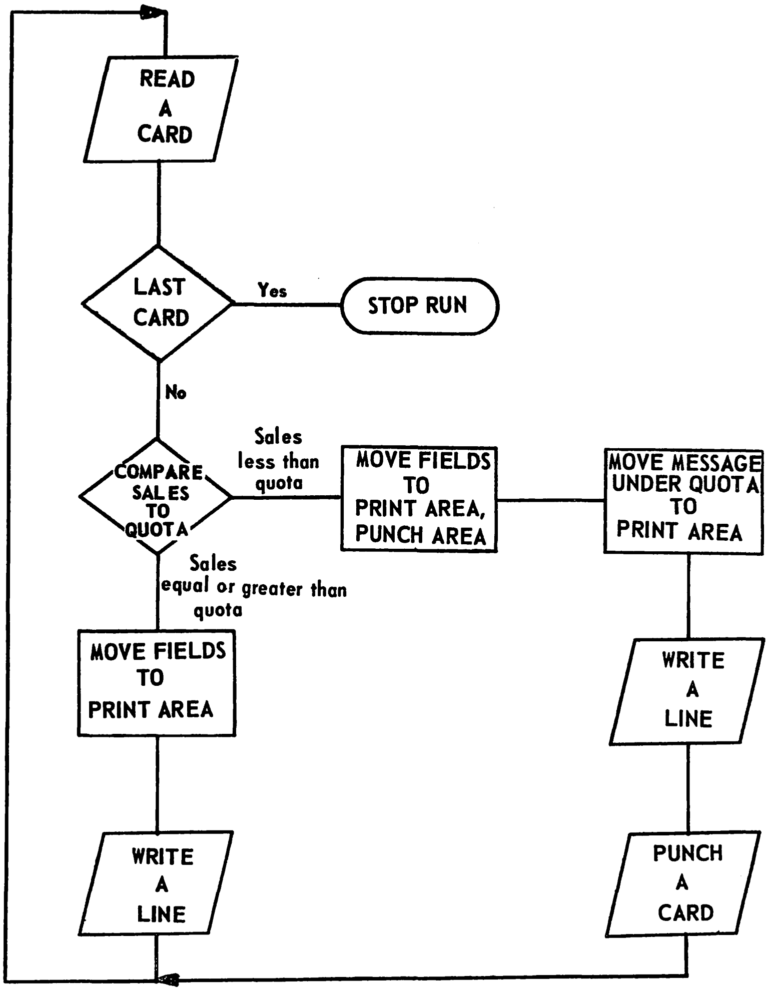
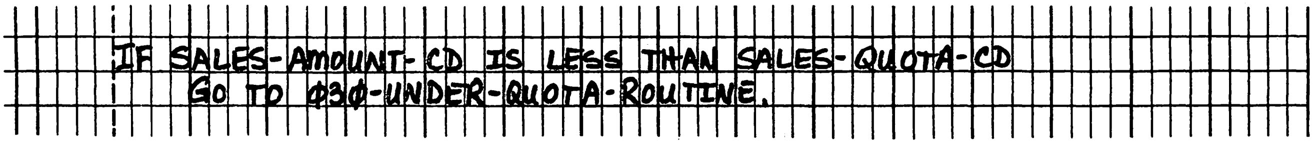

READ A CARD
LAST CARD
Yes
STOP RUN
No
COMPARE SALES TO QUOTA
Sales less than quota
MOVE FIELDS TO PRINT AREA, PUNCH AREA
MOVE MESSAGE UNDER QUOTA TO PRINT AREA
Sales equal or greater than quota
MOVE FIELDS TO PRINT AREA
WRITE A LINE
WRITE A LINE
PUNCH A CARD

Figure 6-1 Flowchart illustrating a Comparing Operation

IF Statement

The Arithmetic Statements, Move Statements, etc., which have been described in previous chapters are often called "Imperative Statements". Imperative Statements direct the computer to perform certain specified actions. They give directions which are specific and unequivocal. In many applications the programmer may desire to instruct the computer to perform operations based upon comparisons of numeric values or other conditions which may occur during execution of the program. Such instructions are called CONDITIONAL STATEMENTS. Conditional Statements in COBOL normally involve the key word IF, followed by the operations to be performed.

The following is an example of a Conditional statement in which a field referenced by SALES-AMOUNT-CD is compared to a field referenced by SALES-QUOTA-CD.



```
IF SALES-AMOUNT-CD IS LESS THAN SALES-QUOTA-CD
    GO TO 030-UNDER-QUOTA-ROUTINE.
```

Figure 6-2 Example of an IF Statement

In the example COBOL statement above, if SALES-AMOUNT-CD is less than SALES-QUOTA-CD the 030-UNDER-QUOTA-ROUTINE is executed. If SALES-AMOUNT-CD is equal or greater than SALES-QUOTA-CD, the next sentence is executed.

Whenever an IF Statement is encountered, it will be evaluated before any other action is taken. In its simplest form, if the condition specified in the IF Statement is found to be true, the program will carry out whatever action is specified immediately following the condition, whereas if the expression is found to be false, the program will execute the next Statement in the COBOL Program.

Format Notation: If Statement

The format of the IF Statement is illustrated in the following diagram.

$$\underline{\texttt{IF}}\ \texttt{condition}\ \left\{\begin{matrix}\texttt{statement-1}\\ \underline{\texttt{NEXT}}\ \underline{\texttt{SENTENCE}}\end{matrix}\right\}\ \underline{\texttt{ELSE}}\ \left\{\begin{matrix}\texttt{statement-2}\\ \underline{\texttt{NEXT}}\ \underline{\texttt{SENTENCE}}\end{matrix}\right\}$$

Figure 6-3 Format Notation – IF Statement

When using the IF Statement in its fullest form,as illustrated in the format notation above, if the condition specified is found to be true, either Statement 1 or NEXT SENTENCE whichever is specified, will be executed. Statement-1 may consist of one or more imperative statements. If NEXT SENTENCE is specified, the program will bypass the remainder of the sentence within the IF Statement and will proceed to the next sentence in the program.

If the condition is found to be false, the program will take no action until it encounters the word ELSE. It will then execute Statement 2 or NEXT SENTENCE, whichever has been specified. It should be noted that the phrase ELSE NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

Additional examples of the IF statement are illustrated below.

Example 1: If Statement

```
IF YEAR-TO-DATE-GROSS IS GREATER THAN 7800 NEXT SENTENCE
    ELSE GO TO FICA-ROUTINE.
```

Figure 6-4 Example 1 – IF Statement

Example 2: If Statement

```
IF COUNTER IS GREATER THAN 5 GO TO RESET-ROUTINE
    ELSE ADD 1 TO COUNTER GO TO UPDATE-ROUTINE.
```

Figure 6-5 Example 2 – IF Statement

TEST CONDITIONS

With the IF statement a "test-condition" is an expression that, taken as a whole, may be either true or false, depending on the circumstances existing when the expression is evaluated. Three basic types of simple test-conditions include:

1. Relation Test
2. Sign Test
3. Class Test

A relation test involves the comparison of two operands, either of which can be a data-name, a literal, or an arithmetic expression. The format of a relation test is illustrated below.

```
{ identifier-1           }
{ literal-1              }  relational-operator
{ arithmetic-expression-1 }

     { identifier-2           }
     { literal-2              }
     { arithmetic-expression-2 }
```

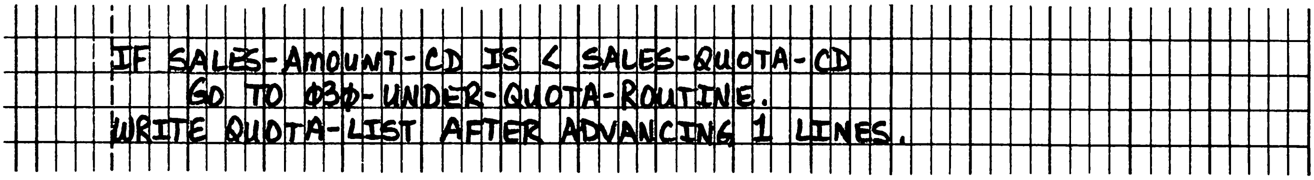
Figure 6-6 Format Notation – Relation Test

The relational-operator specifies the type of comparison to be made in the relation condition. The meaning of the relational operators is shown below.

Relational-operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to

The symbol > is equivalent to the words GREATER THAN. The symbol < is equivalent to the reserved words LESS THAN. The equal sign = is equivalent to the words EQUAL TO.

The following example illustrates a COBOL Statement using the symbol < to represent the COBOL Words LESS THAN. The symbols used in a relation test are normally available on key punches used for punching COBOL programs.



```
IF SALES-AMOUNT-CD IS < SALES-QUOTA-CD
    GO TO 030-UNDER-QUOTA-ROUTINE.
WRITE QUOTA-LIST AFTER ADVANCING 1 LINES.
```

Figure 6-7 Example Relation Test

LOGICAL OPERATORS

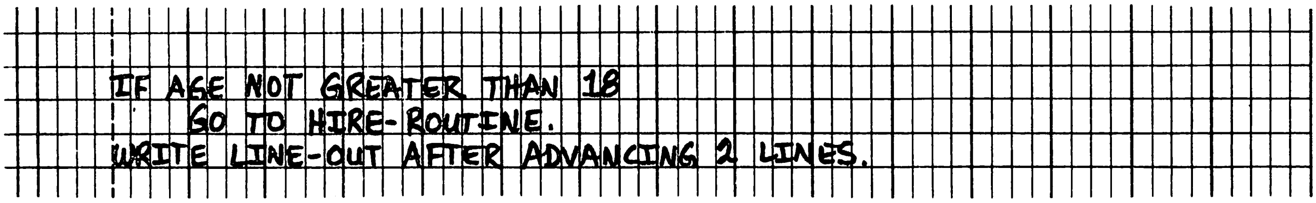
Simple test conditions can be combined with "Logical Operators" according to specified rules to form compound conditions. The logical operators are:

1. NOT
2. OR
3. AND

Logical Operator "NOT"

The word NOT may be used in any simple test-condition to make the relation specify the opposite of what it would express without the word NOT. For example the Statement "IF AGE NOT GREATER THAN 18" is the opposite of the Statement "IF AGE GREATER THAN 18".

The following example illustrates the use of a COBOL Statement incorporating the use of the logical operator "NOT".



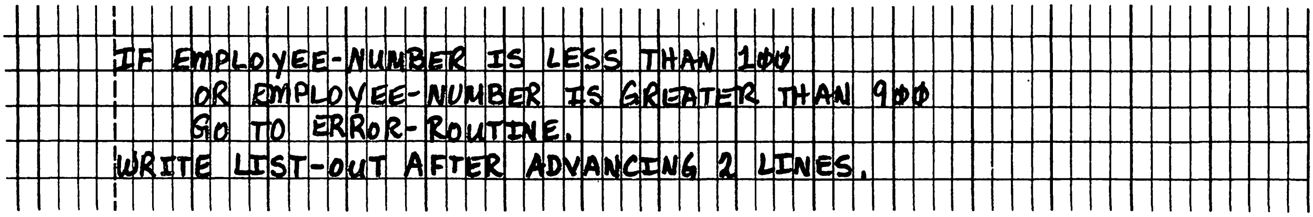
```
IF AGE NOT GREATER THAN 18
    GO TO HIRE-ROUTINE.
WRITE LINE-OUT AFTER ADVANCING 2 LINES.
```

Figure 6-8 Use of "NOT" in an IF Statement

Logical Operator "OR"

Two or more simple conditions combined by AND and/or OR make up a compound condition.

The word OR is used to mean either or both. The following example illustrates the use of a compound condition using OR.



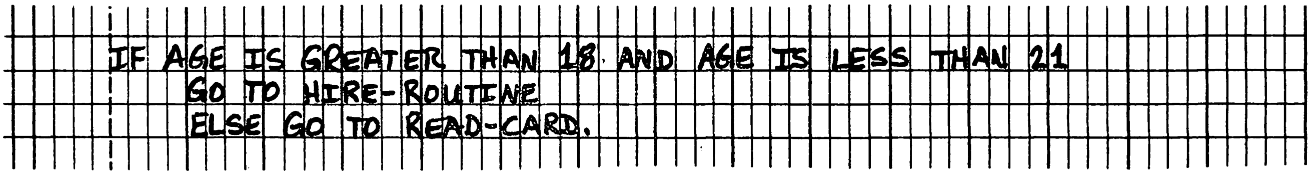
```
IF EMPLOYEE-NUMBER IS LESS THAN 100
    OR EMPLOYEE-NUMBER IS GREATER THAN 900
    GO TO ERROR-ROUTINE.
WRITE LIST-OUT AFTER ADVANCING 2 LINES.
```

Figure 6-9 Use of "OR" in an IF Statement

In this example if either condition is true, the statement "GO TO ERROR-ROUTINE" will be executed. If both conditions are false, the next statement in the COBOL program, the WRITE statement, would be executed.

Logical Operator "AND"

The word AND is used to mean both. The following example illustrates the use of a compound condition using the AND logical operator.



```
IF AGE IS GREATER THAN 18 AND AGE IS LESS THAN 21
    GO TO HIRE-ROUTINE
    ELSE GO TO READ-CARD.
```

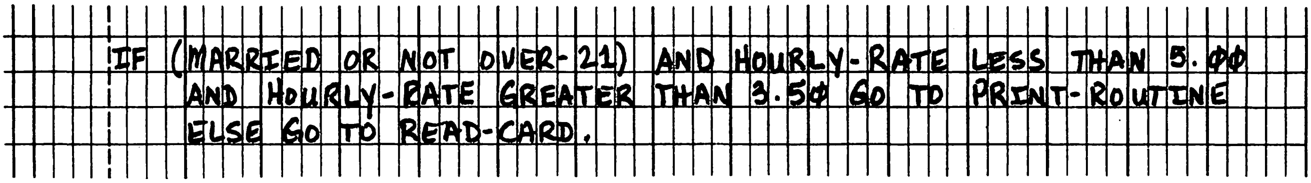
Figure 6-10 Use of "AND" in an IF Statement

In the example of the "and" logical operator above the computer will execute the "GO TO HIRE-ROUTINE" only if both conditions are true.

EVALUATING COMPOUND CONDITIONS

Parentheses may be used to specify the order in which conditions are evaluated. Parentheses must always be paired. Logical evaluation begins with the innermost pair of parentheses and proceeds to the outermost.

The following example illustrates a COBOL Statement requiring the use of the parentheses. In this example a statement is written to search a personnel file for every person married or not over 21 whose pay rate is between 3.50 and 5.00 dollars per hour.



```
IF (MARRIED OR NOT OVER-21) AND HOURLY-RATE LESS THAN 5.00
    AND HOURLY-RATE GREATER THAN 3.50 GO TO PRINT-ROUTINE
    ELSE GO TO READ-CARD.
```

Figure 6-11 Use of Parentheses in a COBOL Statement

In the example above evaluation begins with the condition specified within the parentheses, thus, MARRIED OR NOT OVER-21 would be evaluated as either true or false before the rest of the expression is considered.

If the order of evaluation is not specified by parentheses, the expression is evaluated as follows:

1. AND and its surrounding conditions are evaluated first, starting at the left of the expression and proceeding to the right.
2. OR and its surrounding conditions are then evaluated, also working from left to right.

Thus, in Figure 10-11, if the statement did not contain parentheses, it would be evaluated as MARRIED OR (NOT OVER-21 AND HOURLY-RATE LESS THAN 5.00 AND HOURLY-RATE GREATER THAN 3.50), which is not the desired condition to be tested.

The following chart summarizes some of the basic rules and applications concerning compound conditional expressions.

Compound Conditional Expression	**Interpretation**
MARRIED OR DIVORCED	Either or both of these two conditions must be true for the expression as a whole to be true.
MARRIED AND NOT AGE GREATER THAN 21	The condition MARRIED must be fulfilled and the condition AGE GREATER THAN 21 must *not* be fulfilled for this expression to be true.
A AND (B OR C) OR D	One of the following must be true for the expression to be true: 1. D must be true, or 2. A must be true and either B or C (or both) must be true, or 3. Both the above must be true.
A AND B OR C OR D	If both A and B are true, then the condition of C and D do not matter. If either A or B (or both) is false, then either C or D (or both) must be true for the expression as a whole to be true.

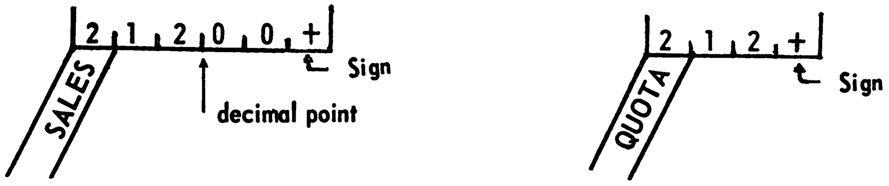
COMPARISON OF NUMERIC ITEMS - (PICTURE 9)

For numeric items, a relation test determines if the value of one of the items is less than, equal to, or greater than the other, regardless of length. When both items are numeric, the comparison is based directly on their algebraic values. Length differences are ignored. For example, 07.6000 is considered to be equal to 0007.60. If no decimal points are indicated, each digit is a separate integer and comparison is made on that basis. For example, 076000 would be considered 76,000, and 000760 would be considered 760 (seven hundred and sixty).

Since zero is considered a unique value, neither positive nor negative, items with zero values are considered to be equal regardless of length, decimal points, and signs.

The following example illustrates the comparison of numeric items.

```
     03  SALES                        PICTURE 999V99.
     03  QUOTA                        PICTURE 999.
```



```
     IF SALES IS EQUAL TO QUOTA GO TO PRINT-ROUTINE.
```

Figure 6-12 Comparison of Numeric Items

In the example above, when SALES is compared to QUOTA the condition existing would be evaluated as true, that is, the fields would be considered equal.

It should be noted that when comparing fields defined as numeric (PICTURE 9), the fields must contain valid plus or minus signs. As the sign of the field is considered in a numeric compare, any value other than a sign in the sign position may result in the termination of the execution of the program.

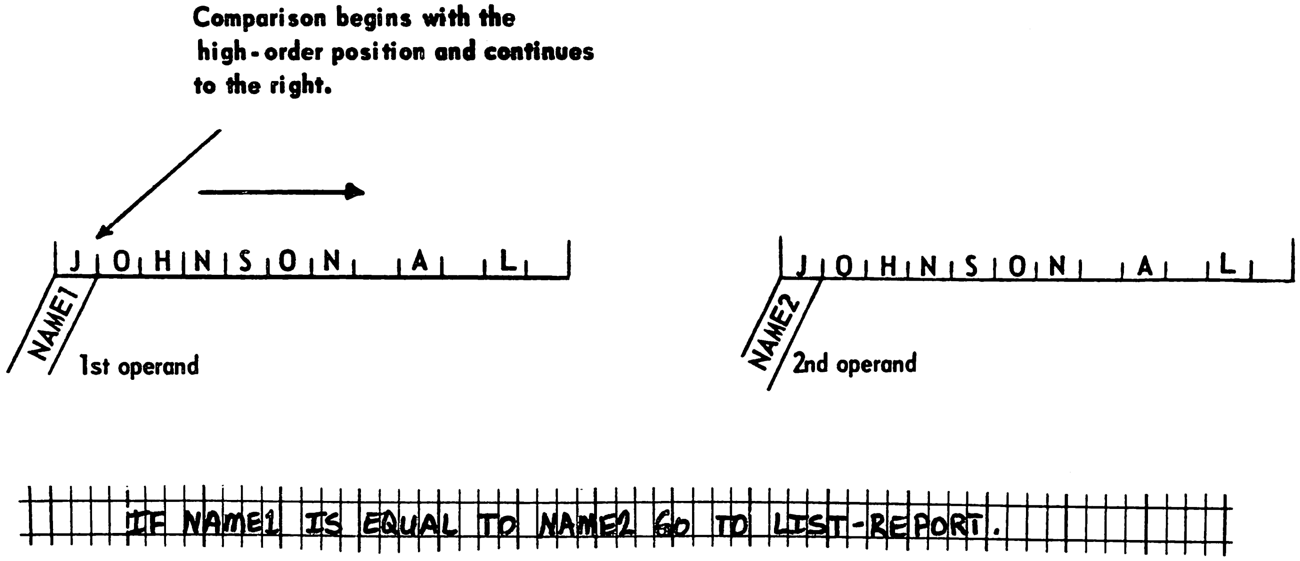
COMPARISON OF NON-NUMERIC ITEMS - (PICTURE X)

For non-numeric items, a comparison results in the determination that one of the items is less than, equal to, or greater than the other, with respect to the binary collating sequence of characters in the character set of the computer.

If non-numeric items are of the same length, the comparison proceeds by comparing characters in each of the fields being referenced, starting from the high-order position and continuing until either a pair of unequal characters or the low-order position of the items are compared.

The following example illustrates the comparison of non-numeric items.

Example 1: Comparison of non-numeric items


Comparison begins with the high-order position and continues to the right.
J O H N S O N A L
NAME1
1st operand
J O H N S O N A L
NAME2
2nd operand

```
IF NAME1 IS EQUAL TO NAME2 GO TO LIST-REPORT.
```

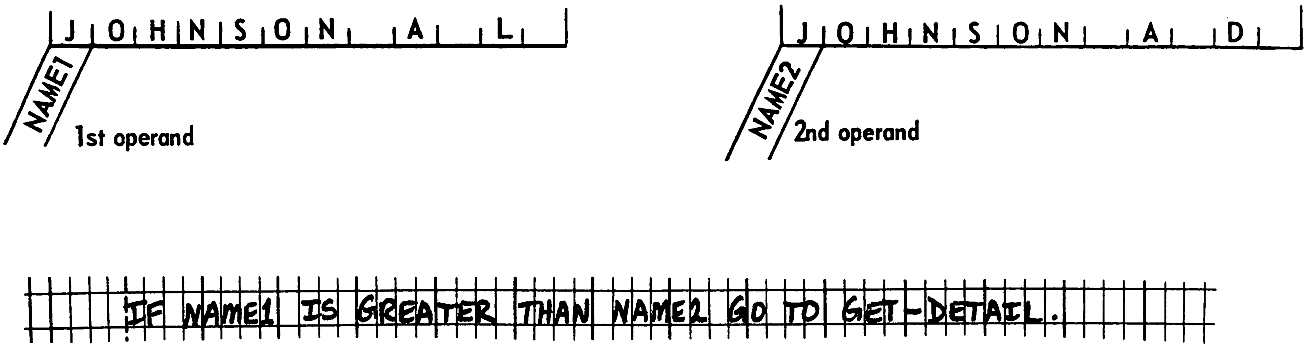
Figure 6-13 Example 1: Comparison of Non-Numeric Items

Note in the example that comparison begins with the high-order position of the fields being referenced.

When comparing non-numeric items and an unequal condition occurs, the first pair of unequal characters encountered is compared for relative position in the collating sequence. The item containing the character that is positioned higher in the collating sequence is the greater item.

The following diagram illustrates the comparison of a non-numeric item in which an unequal condition results.

Example 2: Comparison of non-numeric items



```
IF NAME1 IS GREATER THAN NAME2 GO TO GET-DETAIL.
```

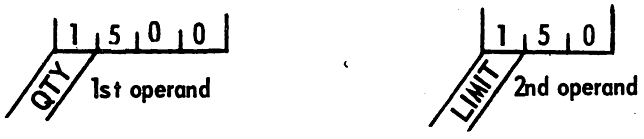
Figure 6-14 Example 2: Comparison of Non-Numeric Items

In the example above, the data in the area referenced by NAME1 is higher in the collating sequence than the data in the area referenced by NAME2, therefore, the condition specified in the IF statement would be evaluated as true, and the statement GO TO GET-DETAIL would be executed next.

If the non-numeric items are of UNEQUAL LENGTH, the shorter field is considered less than the longer field, unless the remainder of the longer item consists solely of spaces in which case the items are considered equal.

The following example illustrates the comparison of non-numeric items in which the fields compared are of unequal length.

Example 3: Comparison of non–numeric items



```
    IF QTY IS EQUAL TO LIMIT GO TO REORDER,
       ELSE GO TO READ-INPUT.
```

Figure 6-15 Example 3: Comparison of Non-Numeric Items

In the example above, the evaluation is false, that is, the fields are considered unequal.

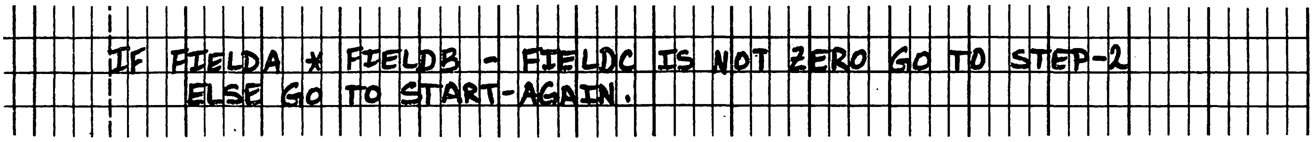
SIGN TEST

The Sign Test is a type of condition test that may be used within an IF Statement to test whether or not the value of a numeric item is less than zero (NEGATIVE), is greater than zero (POSITIVE), or is zero (ZERO). The value zero is considered neither positive or negative. The format notation for a Sign Test is illustrated below.

$$\left\{\begin{array}{l}\texttt{identifier}\\ \texttt{arithmetic-expression}\end{array}\right\}\ \texttt{IS [}\underline{\texttt{NOT}}\texttt{]}\ \left\{\begin{array}{l}\underline{\texttt{POSITIVE}}\\ \underline{\texttt{NEGATIVE}}\\ \underline{\texttt{ZERO}}\end{array}\right\}$$

Figure 6-16 Format Notation – Sign Test

The following example illustrates a COBOL Statement incorporating a Sign Test.



```
IF FIELDA * FIELDB - FIELDC IS NOT ZERO GO TO STEP-2
    ELSE GO TO START-AGAIN.
```

Figure 6-17 Example COBOL Statement – Sign Test

The sign condition test is merely a convenient way of expressing certain conditions that could also be expressed by a simple relational condition.

CLASS TEST

When a Class Test is specified, determination is made as to whether or not an item consists solely of the following:

1. The characters 0 through 9 (NUMERIC).
2. The characters A through Z and space (ALPHABETIC).

The ALPHABETIC test may be performed only on elementary alphabetic or alphanumeric items (items with a Picture of "A" or a Picture of "X").

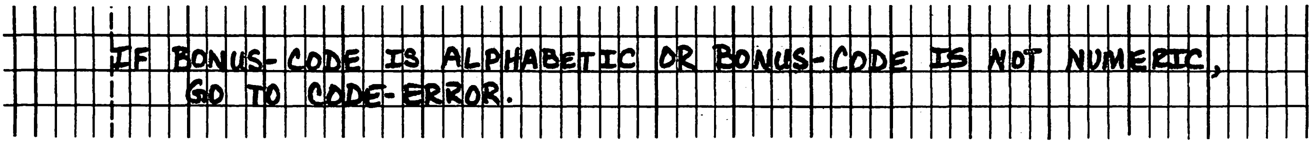
The NUMERIC test may be performed on elementary alphanumeric, internal decimal, or external decimal items (items with a Picture of "9" or a Picture of "X").

The format notation of the Class Test is illustrated below.

```
                      { NUMERIC    }
identifier IS [NOT]   {            }
                      { ALPHABETIC }
```

Figure 6-18 Format Notation – Class Test

The following example illustrates a COBOL Statement using a Class Test.



```
IF BONUS-CODE IS ALPHABETIC OR BONUS-CODE IS NOT NUMERIC,
    GO TO CODE-ERROR.
```

Figure 6-19 Example COBOL Statement using a Class Test

In the sample COBOL statement above, a field identified as BONUS-CODE is tested for the presence of a punch other than a number. If a special character or a letter of the alphabet is contained in the area referenced by BONUS-CODE, the statement GO TO CODE-ERROR is executed.

It should be noted that a field defined as numeric (Picture with 9's) cannot be tested as Alphabetic.

SAMPLE PROBLEM

The following pages contain the documentation for a sample problem designed to illustrate the use of the If Statement. Input consists of Employee Sales Cards containing the Employee Number, Employee Name, a Sales Amount Field, and a Sales Quota Field.

The program is designed to read the Employee Sales Cards, compare the Sales Amount punched on the card to the Sales Quota punched on the same card. If the Sales Amount is equal to, or greater than, the Sales Quota, a line is to be printed. If the Sales Amount is less than the Sales Quota, a line is to be printed with the identifying message "UNDER QUOTA". In addition, a card is to be punched containing all the fields on the original input card if sales are "under quota".

Input:

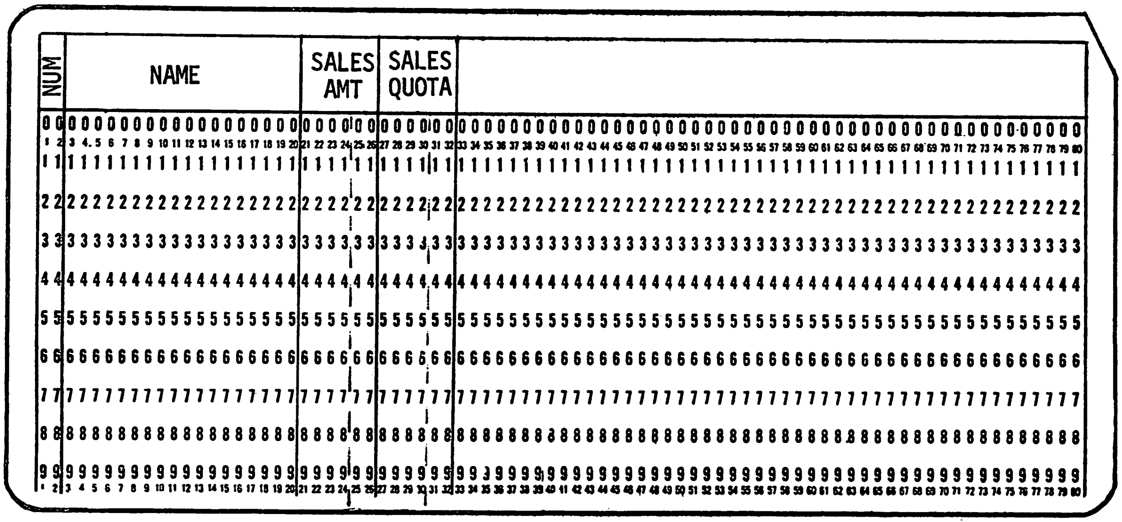
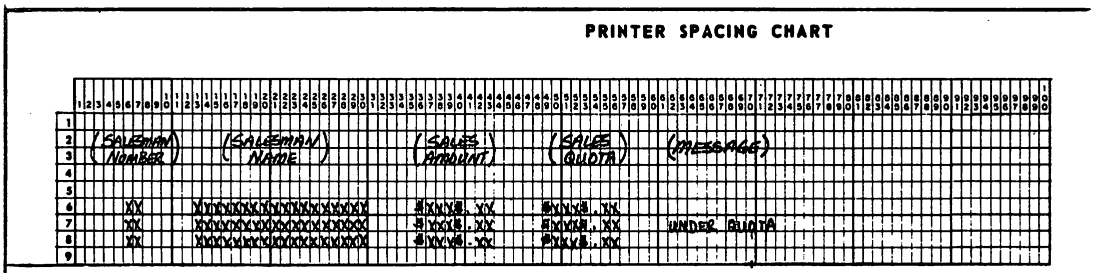

NUM
NAME
SALES AMT
SALES QUOTA

Figure 6-20 Input

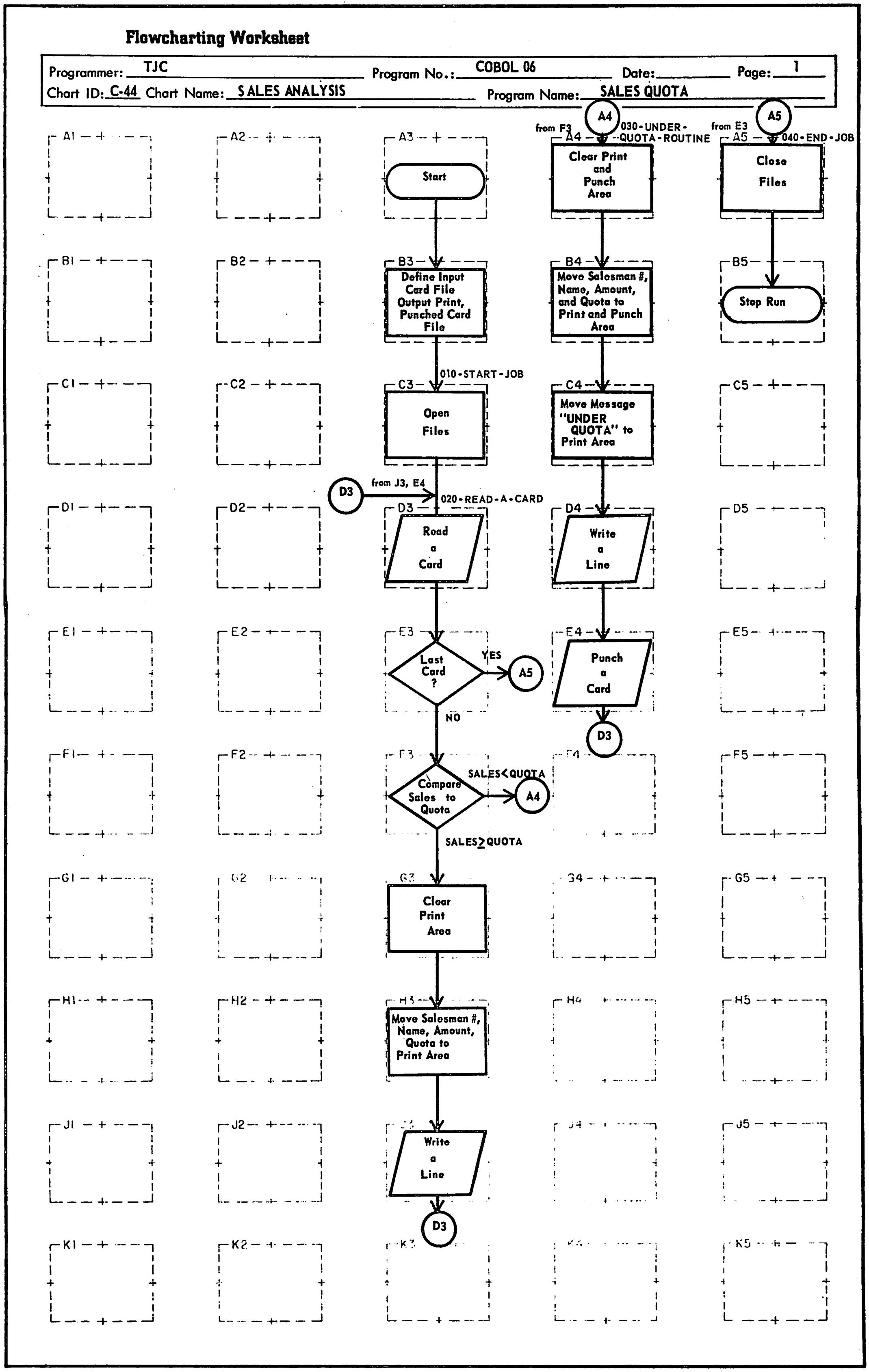
Output:


PRINTER SPACING CHART
(SALESMAN NUMBER)
(SALESMAN NAME)
(SALES AMOUNT)
(SALES QUOTA)
(MESSAGE)
XX
XXXXXXXXXXXXXXXXXXXX
XXX.XX
XXX.XX
UNDER QUOTA

```
01    LEMONS, ERIC         $1000.00   $1200.00    UNDER QUOTA
02    ABRAMS, DON          $1000.00   $1000.00
03    WILLIAMS, BARBARA     $200.00   $1500.00    UNDER QUOTA
04    MEEK, DAVE            $500.00    $800.00    UNDER QUOTA
05    STERLING, TINA       $9999.99   $9999.99
06    BLACKARD, LINDA          $.00    $300.00    UNDER QUOTA
```

Figure 6-21 Output

FLOWCHART



Flowcharting Worksheet

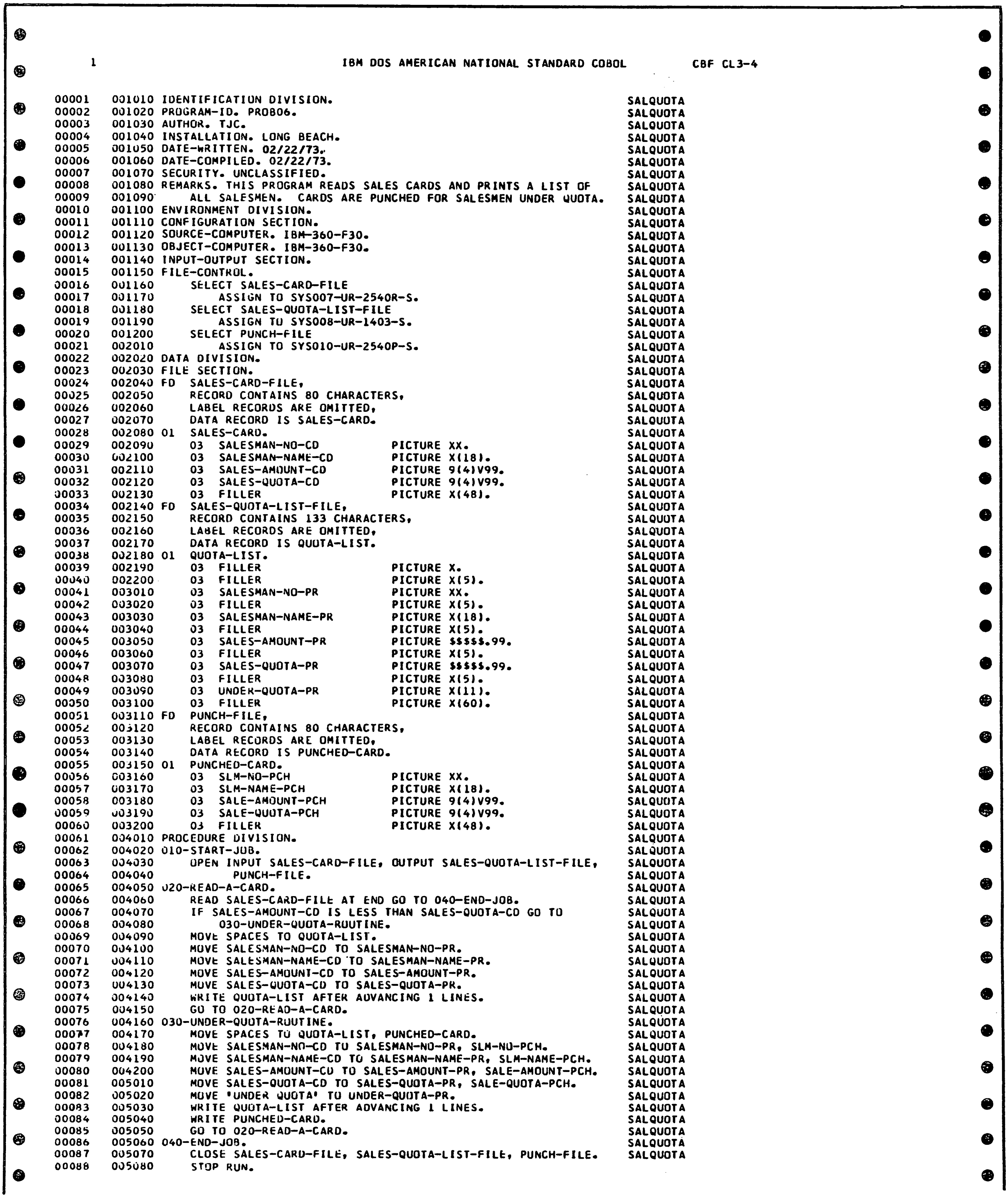
Programmer: TJC Program No.: COBOL 06 Date: Page: 1

Chart ID: C-44 Chart Name: SALES ANALYSIS Program Name: SALES QUOTA

Start
Define Input Card File Output Print, Punched Card File
010-START-JOB
Open Files
D3
from J3, E4
020-READ-A-CARD
Read a Card
Last Card ?
YES
A5
NO
Compare Sales to Quota
SALES<QUOTA
A4
SALES≥QUOTA
Clear Print Area
Move Salesman #, Name, Amount, Quota to Print Area
Write a Line
D3
A4
from F3
030-UNDER-QUOTA-ROUTINE
Clear Print and Punch Area
Move Salesman #, Name, Amount, and Quota to Print and Punch Area
Move Message "UNDER QUOTA" to Print Area
Write a Line
Punch a Card
D3
A5
from E3
040-END-JOB
Close Files
Stop Run
A1
A2
A3
A4
A5
B1
B2
B3
B4
B5
C1
C2
C3
C4
C5
D1
D2
D3
D4
D5
E1
E2
E3
E4
E5
F1
F2
F3
F4
F5
G1
G2
G3
G4
G5
H1
H2
H3
H4
H5
J1
J2
J3
J4
J5
K1
K2
K3
K4
K5

Figure 6-22 Flowchart

Source Listing



```
      1                                IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4

00001   001010 IDENTIFICATION DIVISION.                                         SALQUOTA
00002   001020 PROGRAM-ID. PROB06.                                              SALQUOTA
00003   001030 AUTHOR. TJC.                                                     SALQUOTA
00004   001040 INSTALLATION. LONG BEACH.                                        SALQUOTA
00005   001050 DATE-WRITTEN. 02/22/73.                                          SALQUOTA
00006   001060 DATE-COMPILED. 02/22/73.                                         SALQUOTA
00007   001070 SECURITY. UNCLASSIFIED.                                          SALQUOTA
00008   001080 REMARKS. THIS PROGRAM READS SALES CARDS AND PRINTS A LIST OF     SALQUOTA
00009   001090     ALL SALESMEN.  CARDS ARE PUNCHED FOR SALESMEN UNDER QUOTA.   SALQUOTA
00010   001100 ENVIRONMENT DIVISION.                                            SALQUOTA
00011   001110 CONFIGURATION SECTION.                                           SALQUOTA
00012   001120 SOURCE-COMPUTER. IBM-360-F30.                                    SALQUOTA
00013   001130 OBJECT-COMPUTER. IBM-360-F30.                                    SALQUOTA
00014   001140 INPUT-OUTPUT SECTION.                                            SALQUOTA
00015   001150 FILE-CONTROL.                                                    SALQUOTA
00016   001160     SELECT SALES-CARD-FILE                                       SALQUOTA
00017   001170         ASSIGN TO SYS007-UR-2540R-S.                             SALQUOTA
00018   001180     SELECT SALES-QUOTA-LIST-FILE                                 SALQUOTA
00019   001190         ASSIGN TO SYS008-UR-1403-S.                              SALQUOTA
00020   001200     SELECT PUNCH-FILE                                            SALQUOTA
00021   002010         ASSIGN TO SYS010-UR-2540P-S.                             SALQUOTA
00022   002020 DATA DIVISION.                                                   SALQUOTA
00023   002030 FILE SECTION.                                                    SALQUOTA
00024   002040 FD  SALES-CARD-FILE,                                             SALQUOTA
00025   002050     RECORD CONTAINS 80 CHARACTERS,                               SALQUOTA
00026   002060     LABEL RECORDS ARE OMITTED,                                   SALQUOTA
00027   002070     DATA RECORD IS SALES-CARD.                                   SALQUOTA
00028   002080 01  SALES-CARD.                                                  SALQUOTA
00029   002090     03  SALESMAN-NO-CD          PICTURE XX.                      SALQUOTA
00030   002100     03  SALESMAN-NAME-CD        PICTURE X(18).                   SALQUOTA
00031   002110     03  SALES-AMOUNT-CD         PICTURE 9(4)V99.                 SALQUOTA
00032   002120     03  SALES-QUOTA-CD          PICTURE 9(4)V99.                 SALQUOTA
00033   002130     03  FILLER                  PICTURE X(48).                   SALQUOTA
00034   002140 FD  SALES-QUOTA-LIST-FILE,                                       SALQUOTA
00035   002150     RECORD CONTAINS 133 CHARACTERS,                              SALQUOTA
00036   002160     LABEL RECORDS ARE OMITTED,                                   SALQUOTA
00037   002170     DATA RECORD IS QUOTA-LIST.                                   SALQUOTA
00038   002180 01  QUOTA-LIST.                                                  SALQUOTA
00039   002190     03  FILLER                  PICTURE X.                       SALQUOTA
00040   002200     03  FILLER                  PICTURE X(5).                    SALQUOTA
00041   003010     03  SALESMAN-NO-PR          PICTURE XX.                      SALQUOTA
00042   003020     03  FILLER                  PICTURE X(5).                    SALQUOTA
00043   003030     03  SALESMAN-NAME-PR        PICTURE X(18).                   SALQUOTA
00044   003040     03  FILLER                  PICTURE X(5).                    SALQUOTA
00045   003050     03  SALES-AMOUNT-PR         PICTURE $$$$$.99.                SALQUOTA
00046   003060     03  FILLER                  PICTURE X(5).                    SALQUOTA
00047   003070     03  SALES-QUOTA-PR          PICTURE $$$$$.99.                SALQUOTA
00048   003080     03  FILLER                  PICTURE X(5).                    SALQUOTA
00049   003090     03  UNDER-QUOTA-PR          PICTURE X(11).                   SALQUOTA
00050   003100     03  FILLER                  PICTURE X(60).                   SALQUOTA
00051   003110 FD  PUNCH-FILE,                                                  SALQUOTA
00052   003120     RECORD CONTAINS 80 CHARACTERS,                               SALQUOTA
00053   003130     LABEL RECORDS ARE OMITTED,                                   SALQUOTA
00054   003140     DATA RECORD IS PUNCHED-CARD.                                 SALQUOTA
00055   003150 01  PUNCHED-CARD.                                                SALQUOTA
00056   003160     03  SLM-NO-PCH              PICTURE XX.                      SALQUOTA
00057   003170     03  SLM-NAME-PCH            PICTURE X(18).                   SALQUOTA
00058   003180     03  SALE-AMOUNT-PCH         PICTURE 9(4)V99.                 SALQUOTA
00059   003190     03  SALE-QUOTA-PCH          PICTURE 9(4)V99.                 SALQUOTA
00060   003200     03  FILLER                  PICTURE X(48).                   SALQUOTA
00061   004010 PROCEDURE DIVISION.                                              SALQUOTA
00062   004020 010-START-JOB.                                                   SALQUOTA
00063   004030     OPEN INPUT SALES-CARD-FILE, OUTPUT SALES-QUOTA-LIST-FILE,    SALQUOTA
00064   004040         PUNCH-FILE.                                              SALQUOTA
00065   004050 020-READ-A-CARD.                                                 SALQUOTA
00066   004060     READ SALES-CARD-FILE AT END GO TO 040-END-JOB.               SALQUOTA
00067   004070     IF SALES-AMOUNT-CD IS LESS THAN SALES-QUOTA-CD GO TO         SALQUOTA
00068   004080         030-UNDER-QUOTA-ROUTINE.                                 SALQUOTA
00069   004090     MOVE SPACES TO QUOTA-LIST.                                   SALQUOTA
00070   004100     MOVE SALESMAN-NO-CD TO SALESMAN-NO-PR.                       SALQUOTA
00071   004110     MOVE SALESMAN-NAME-CD TO SALESMAN-NAME-PR.                   SALQUOTA
00072   004120     MOVE SALES-AMOUNT-CD TO SALES-AMOUNT-PR.                     SALQUOTA
00073   004130     MOVE SALES-QUOTA-CD TO SALES-QUOTA-PR.                       SALQUOTA
00074   004140     WRITE QUOTA-LIST AFTER ADVANCING 1 LINES.                    SALQUOTA
00075   004150     GO TO 020-READ-A-CARD.                                       SALQUOTA
00076   004160 030-UNDER-QUOTA-ROUTINE.                                         SALQUOTA
00077   004170     MOVE SPACES TO QUOTA-LIST, PUNCHED-CARD.                     SALQUOTA
00078   004180     MOVE SALESMAN-NO-CD TO SALESMAN-NO-PR, SLM-NO-PCH.           SALQUOTA
00079   004190     MOVE SALESMAN-NAME-CD TO SALESMAN-NAME-PR, SLM-NAME-PCH.     SALQUOTA
00080   004200     MOVE SALES-AMOUNT-CD TO SALES-AMOUNT-PR, SALE-AMOUNT-PCH.    SALQUOTA
00081   005010     MOVE SALES-QUOTA-CD TO SALES-QUOTA-PR, SALE-QUOTA-PCH.       SALQUOTA
00082   005020     MOVE 'UNDER QUOTA' TO UNDER-QUOTA-PR.                        SALQUOTA
00083   005030     WRITE QUOTA-LIST AFTER ADVANCING 1 LINES.                    SALQUOTA
00084   005040     WRITE PUNCHED-CARD.                                          SALQUOTA
00085   005050     GO TO 020-READ-A-CARD.                                       SALQUOTA
00086   005060 040-END-JOB.                                                     SALQUOTA
00087   005070     CLOSE SALES-CARD-FILE, SALES-QUOTA-LIST-FILE, PUNCH-FILE.    SALQUOTA
00088   005080     STOP RUN.
```

Figure 6-23 COBOL Source Listing

CHAPTER 6

PROGRAMMING ASSIGNMENT 1

INSTRUCTIONS

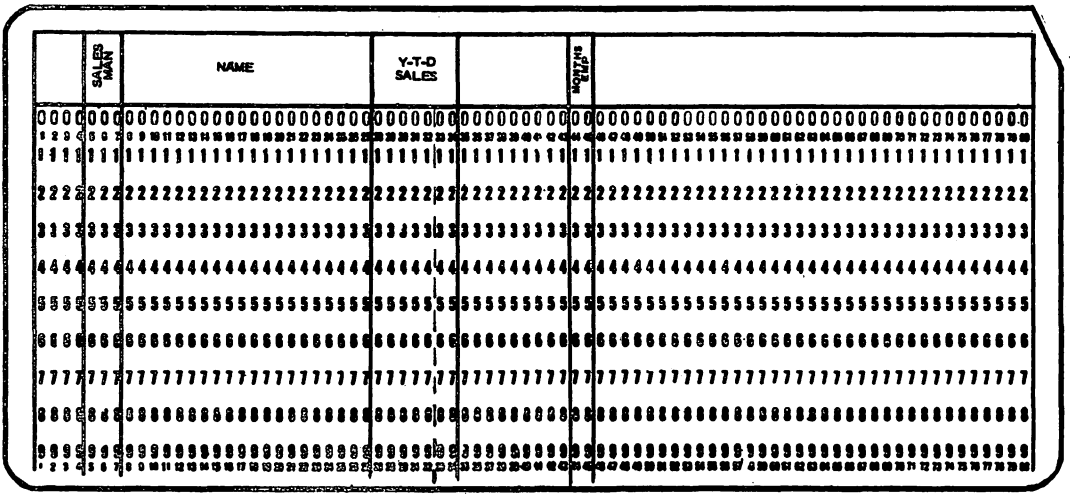
From the Sales cards illustrated below, a Sales Quota Report is to be prepared. This report is to list all employees employed six months or more that have a Y-T-D Sales below $6,000.00. Employees employed less than six months or with Sales $6,000.00 or more are NOT to be included on the report.

The Salesman Number, Salesman Name, Y-T-D Sales, and the Months Employed are to be printed on the report. The Amount below quota is to be calculated and printed on the right side of each report.

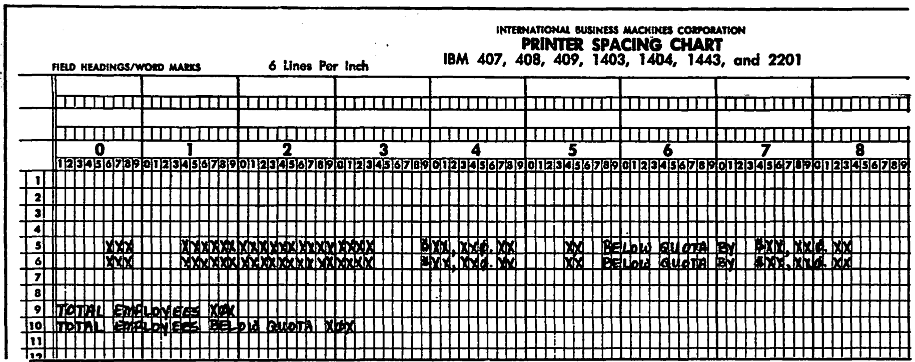
A "final total line" is to be printed indicating the total number of employees, and the total number of employees below quota.

Test Data is contained in the appendices.

INPUT: Sales Card


SALES MAN
NAME
Y-T-D SALES
MONTHS EMP

OUTPUT: Sales Quota Report


INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, and 2201
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX XX BELOW QUOTA BY $XX,XX0.XX
XXX XXXXXXXXXXXXXXXXXXXX $XX,XX0.XX XX BELOW QUOTA BY $XX,XX0.XX
TOTAL EMPLOYEES XXX
TOTAL EMPLOYEES BELOW QUOTA XXX

```
      293      CARTER, A.                 $ 2,235.00       08  BELOW QUOTA BY  $ 3,765.00
      856      SELDA, R.                  $ 5,900.00       06  BELOW QUOTA BY  $   100.00

TOTAL EMPLOYEES    6
TOTAL EMPLOYEES BELOW QUOTA    2
```

CHAPTER 6

PROGRAMMING ASSIGNMENT 2

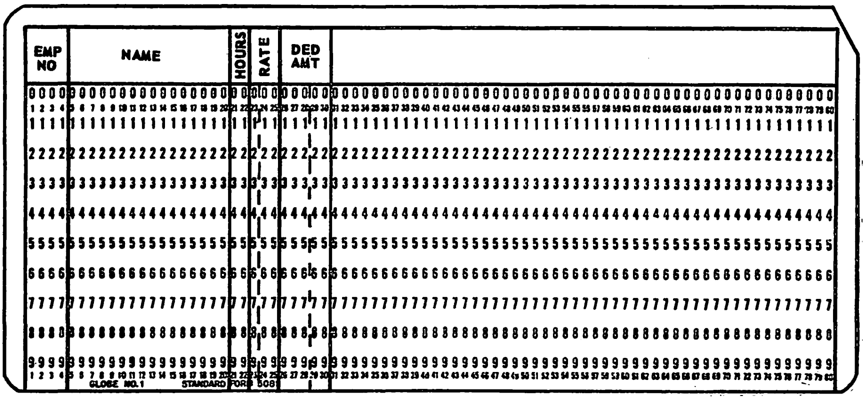
INSTRUCTIONS

Write the COBOL program to prepare a Weekly Payroll Report. Employees are to receive time and one-half for all work in excess of 40 hours. For example, an employee working 42 hours at $2.00 per hour would be paid for 40 hours work at $2.00 per hour and 2 hours work at $3.00 per hour.

Test Data is contained on page 6.20.

INPUT: Payroll Cards

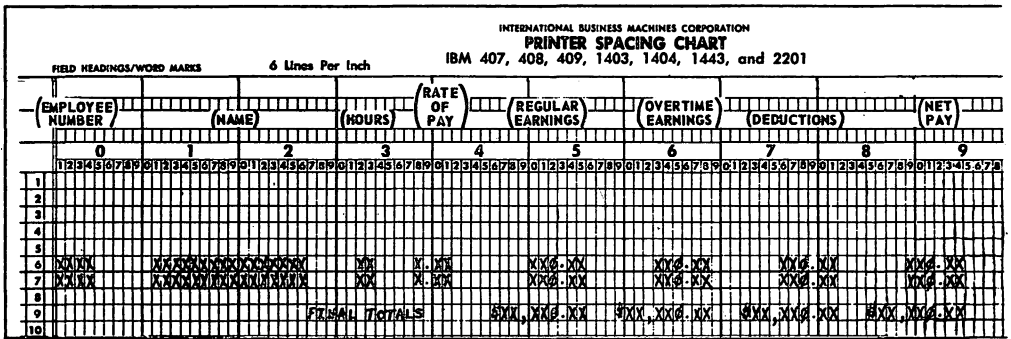
Input is to consist of Payroll cards containing the Employee Number, Employee Name, Hours Worked, Rate of Pay and Deductions. The format of the card is illustrated below.



EMP NO
NAME
HOURS
RATE
DED AMT
GLOBE NO. 1
STANDARD FORM 5081

OUTPUT: Payroll Report

Output is to consist of a Payroll Report. The report is to contain the Employee Number, Employee Name, Total Hours Worked, Regular Rate of Pay, Regular Earnings, Overtime Earnings, Deductions, and Net Pay. Regular Earnings added to Overtime Earnings minus the Deductions will give the Net Pay. Final Totals are to be taken of Regular Earnings, Overtime Earnings, Deductions, and Net Pay.



INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, and 2201
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
EMPLOYEE NUMBER
NAME
HOURS
RATE OF PAY
REGULAR EARNINGS
OVERTIME EARNINGS
DEDUCTIONS
NET PAY
XXXX
XXXXXXXXXXXXXXXXXX
XX
X.XX
XX0.XX
XX0.XX
XX0.XX
XX0.XX
FINAL TOTALS
$XX,XX0.XX
$XX,XX0.XX
$XX,XX0.XX
$XX,XX0.XX

```
1539    ALL, WILLIAM        42    2.00        80.00        6.00       10.00        76.00
1901    AKER, J AMES        30    2.50        75.00         .00        7.00        68.00
3343    BRYANT, ANDREW      45    3.00       120.00       22.50       20.00       122.50
                                               .            .           .            .
                                               .            .           .            .
                                               .            .           .            .
                    FINAL TOTALS         $27,650.00     $350.00   $4,000.00   $24,000.00
```

TEST DATA – ASSIGNMENT NO. 2

Emp No (col 1-4)	Name (col 5-20)	Hours (col 21-22)	Rate (col 23-25)	Deductions (col 26-30)
1539	All, William	42	200	01000
1901	Aker, James	30	250	00700
3343	Bryant, Andrew	45	300	02000
4439	Clayborne, Walt	59	375	04200
5593	Ellisin, William	20	270	00000
7894	Gelden, Robert	48	580	00500

CHAPTER 7

CONDITION-NAMES . . . GO TO OPTIONS

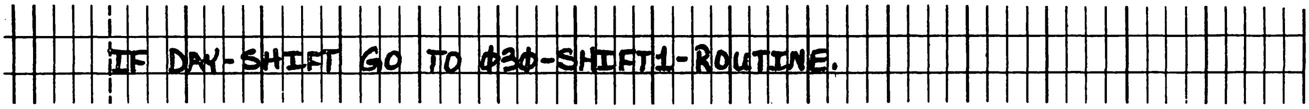
INTRODUCTION

The great power of a computer system rests with the ability of programming languages associated with the system to compare numbers or letters of the alphabet and perform alternative functions based upon the results of these comparisons. The previous chapter introduced some of the capabilities of COBOL in performing comparing type operations. Several additional techniques are useful to the COBOL programmer. These techniques include the use of CONDITION-NAMES, and the use of the "GO TO . . . DEPENDING" Statement.

In many applications records will be identified by a special control punch or code punched onto the card. For example, Employee Cards could be identified by a "1" punch if the employee works on the "day shift", a "3" punch if the employee works on the "night shift", and a "5" punch if the employee works on the "grave yard shift". These codes are normally assigned by the systems analyst at the time the procedure is initially designed.

CONDITION-NAMES

Condition Names are merely symbolic names that are related in the COBOL Program to the control punches or codes on the input data. This technique allows reference to a name rather than a number in the Procedure Division. For example, in the Procedure Division rather than writing "IF CODE-FIELD IS EQUAL TO 1", the use of a Condition Name would allow the statement "IF DAY-SHIFT" (DAY-SHIFT being equated to "1").



```
IF DAY-SHIFT GO TO 030-SHIFT1-ROUTINE.
```

Figure 7-1 Example of Condition-Names used in the Procedure Division

LEVEL 88 ENTRIES

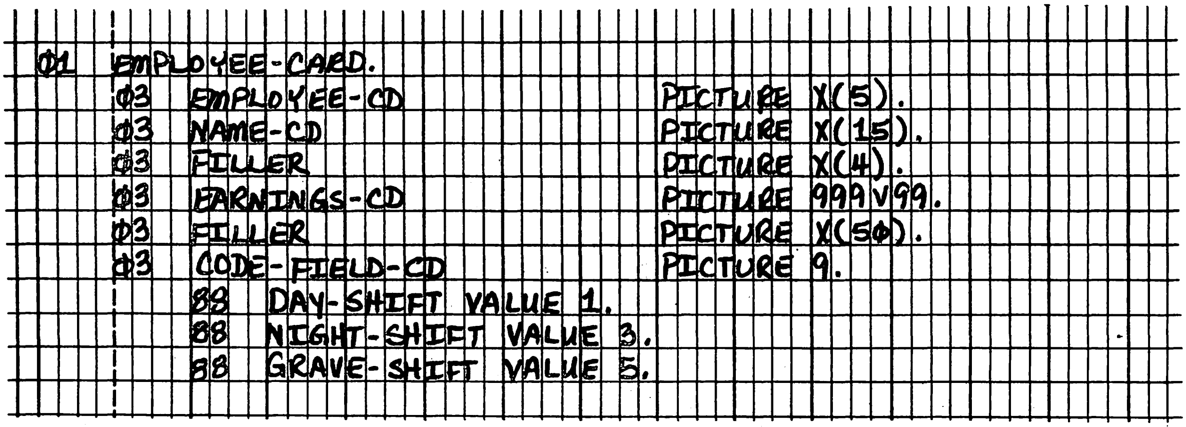
Level Number 88 is used to define a Condition Name. The general format is illustrated below.

```
88 condition-name VALUE Clause.
```

Figure 7-2 Format Notation – Level 88 Entries

The following example illustrates a segment of a COBOL program utilizing CONDITION-NAMES.

This segment of the Data Division describes an input data card containing an Employee Number Field, an Employee Name Field, an Earnings Field, and a Code Field. A "1" punch in the Code Field indicates that the employee works on the Day Shift. A "3" punch in the Code Field indicates that the employee works on the Night Shift, and a "5" punch in the Code Field indicates that the employee works on the Grave Shift. Note how this is specified by means of the Condition-Names and the use of the 88 level.

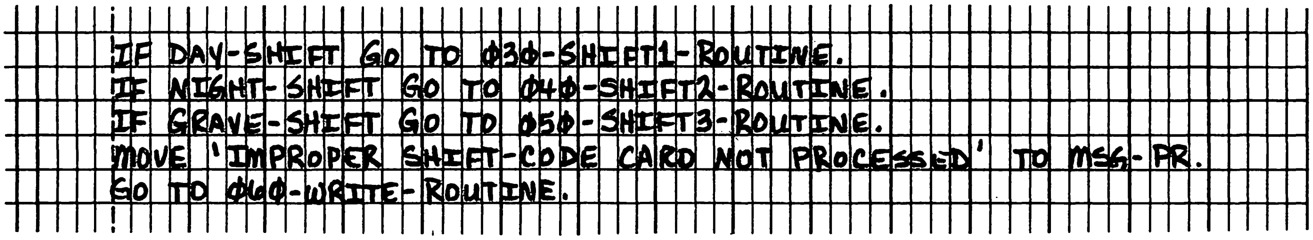


```
01  EMPLOYEE-CARD.
    03  EMPLOYEE-CD                 PICTURE X(5).
    03  NAME-CD                     PICTURE X(15).
    03  FILLER                      PICTURE X(4).
    03  EARNINGS-CD                 PICTURE 999V99.
    03  FILLER                      PICTURE X(50).
    03  CODE-FIELD-CD               PICTURE 9.
        88  DAY-SHIFT VALUE 1.
        88  NIGHT-SHIFT VALUE 3.
        88  GRAVE-SHIFT VALUE 5.
```

Figure 7-3 Example of Level 88 Entries in the Data Division

When using Level 88 entries in the Data Division, each entry must be preceded by either an elementary item or another level 88 entry (in the case of several consecutive condition–names pertaining to an elementary item).

The following is a segment of the Procedure Division in which reference is made to the Condition-Names specified in the Data Division.



```
IF DAY-SHIFT GO TO 030-SHIFT1-ROUTINE.
IF NIGHT-SHIFT GO TO 040-SHIFT2-ROUTINE.
IF GRAVE-SHIFT GO TO 050-SHIFT3-ROUTINE.
MOVE 'IMPROPER SHIFT-CODE CARD NOT PROCESSED' TO MSG-PR.
GO TO 060-WRITE-ROUTINE.
```

Figure 7-4 Example of the use of Condition-Names

From the COBOL entries it can be seen that if the employee works on the "DAY-SHIFT" the 030-SHIFT1-ROUTINE will be executed. If the employee works on "NIGHT-SHIFT" the 040-SHIFT2-ROUTINE will be executed, and that if the employee works on "GRAVE-SHIFT" the 050-SHIFT3-ROUTINE will be executed.

In the example above note that if there is not a "1", punch, a "3" punch, or a "5" punch in the Code Field as specified by the Level 88 entries in the Data Division, the Move Statement is executed. This Statement moves an error message to the output line for printing on the report.

SAMPLE PROBLEM:

The following problem illustrates the complete documentation for an application using Level 88 entries. In this problem the employee's total pay is calculated by adding the Regular Earnings Field (punched on each employee's card) to any "bonus pay" received.

If there is a "1" punch in the Code Field of the Employee Earnings Card, it indicates that the employee works on the Day Shift and is to receive no "bonus pay".

If there is a "3" punch in the card it indicates that the employee works on the Night Shift and is to receive $5.00 in "bonus pay".

If there is a "5" punch in the card it indicates that the employee works on the Grave Yard Shift and is to receive $10.00 in "bonus pay".

In the sample program on page 7.8 note the use of the BLANK clause (Line 52). The BLANK WHEN ZERO clause specifies that the item being described is filled with spaces whenever the value of the item is zero. The BLANK clause may only be used for report items at an elementary level.

The input, output, flowchart and COBOL program for the sample problem illustrating the use of Level 88 entries are contained on the following pages.

Input

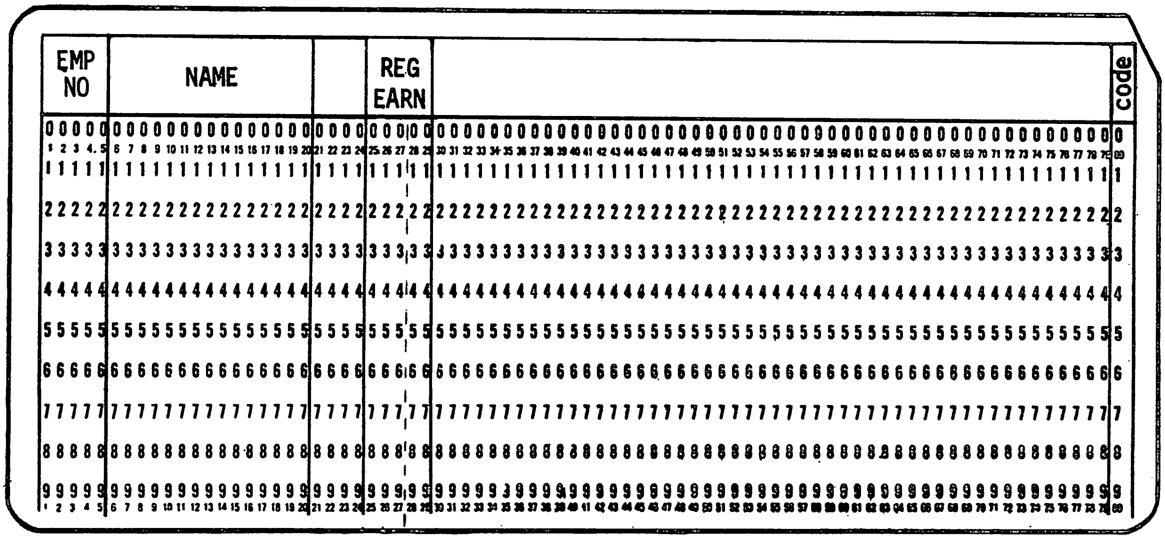
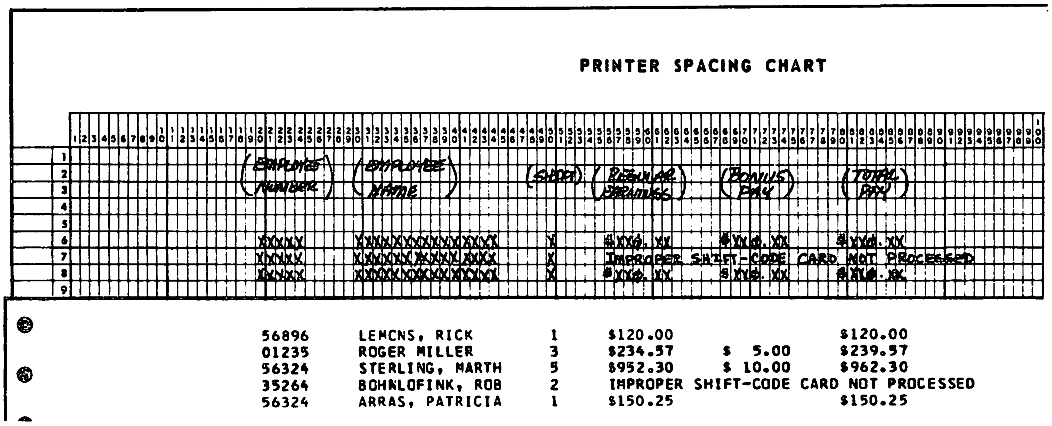

EMP NO
NAME
REG EARN
code

Figure 7-5 Input

Output


PRINTER SPACING CHART
(EMPLOYEE NUMBER)
(EMPLOYEE NAME)
(SHIFT)
(REGULAR EARNINGS)
(BONUS PAY)
(TOTAL PAY)
XXXXX XXXXXXXXXXXXXXX X $XX0.XX $XX0.XX $XX0.XX
XXXXX XXXXXXXXXXXXXXX X IMPROPER SHIFT-CODE CARD NOT PROCESSED
XXXXX XXXXXXXXXXXXXXX X $XX0.XX $XX0.XX $XX0.XX

```
56896     LEMCNS, RICK        1    $120.00                 $120.00
01235     ROGER MILLER        3    $234.57     $  5.00     $239.57
56324     STERLING, MARTH     5    $952.30     $ 10.00     $962.30
35264     BOHNLOFINK, ROB     2    IMPROPER SHIFT-CODE CARD NOT PROCESSED
56324     ARRAS, PATRICIA     1    $150.25                 $150.25
```

Figure 7-6 Output

Flowchart

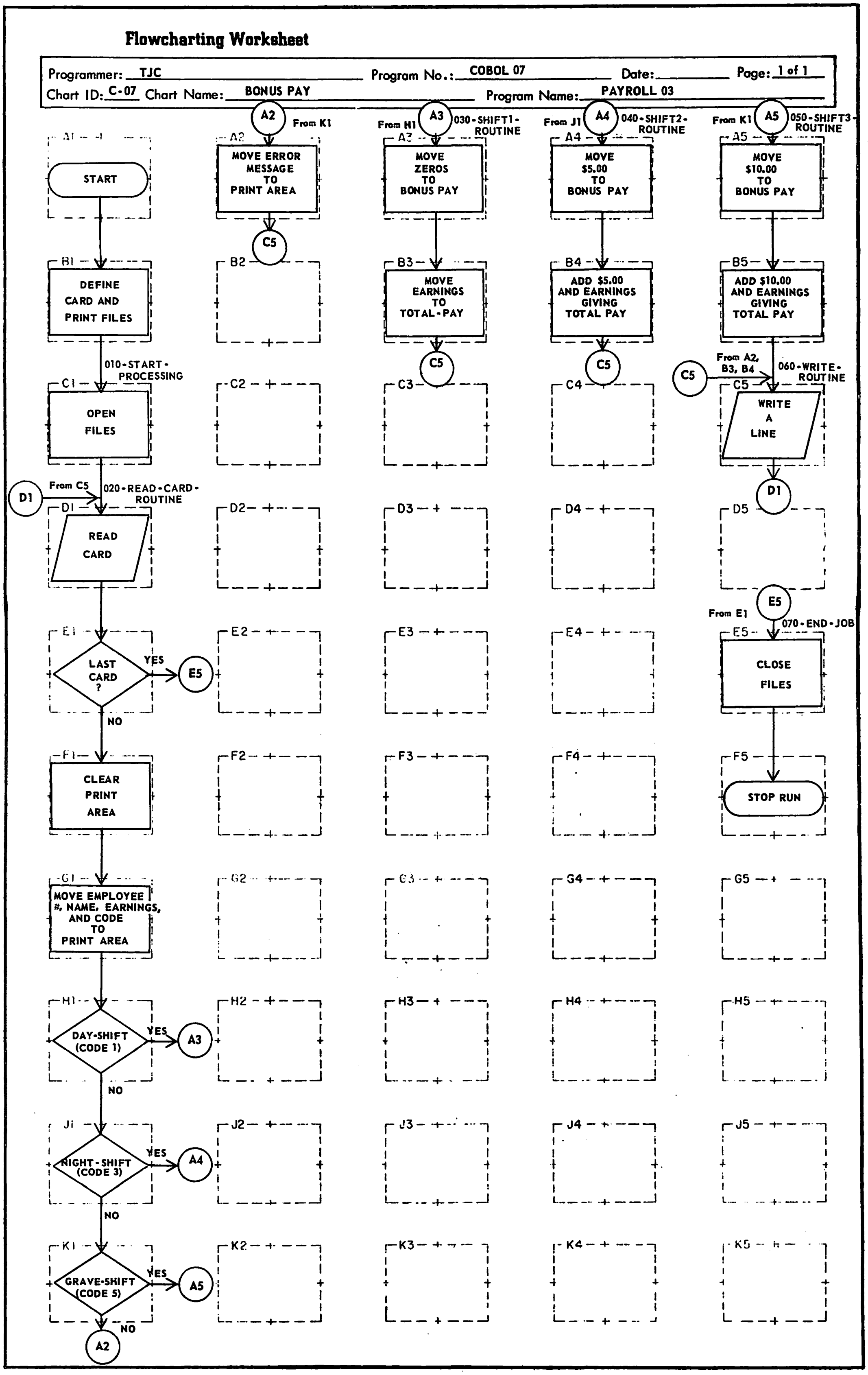
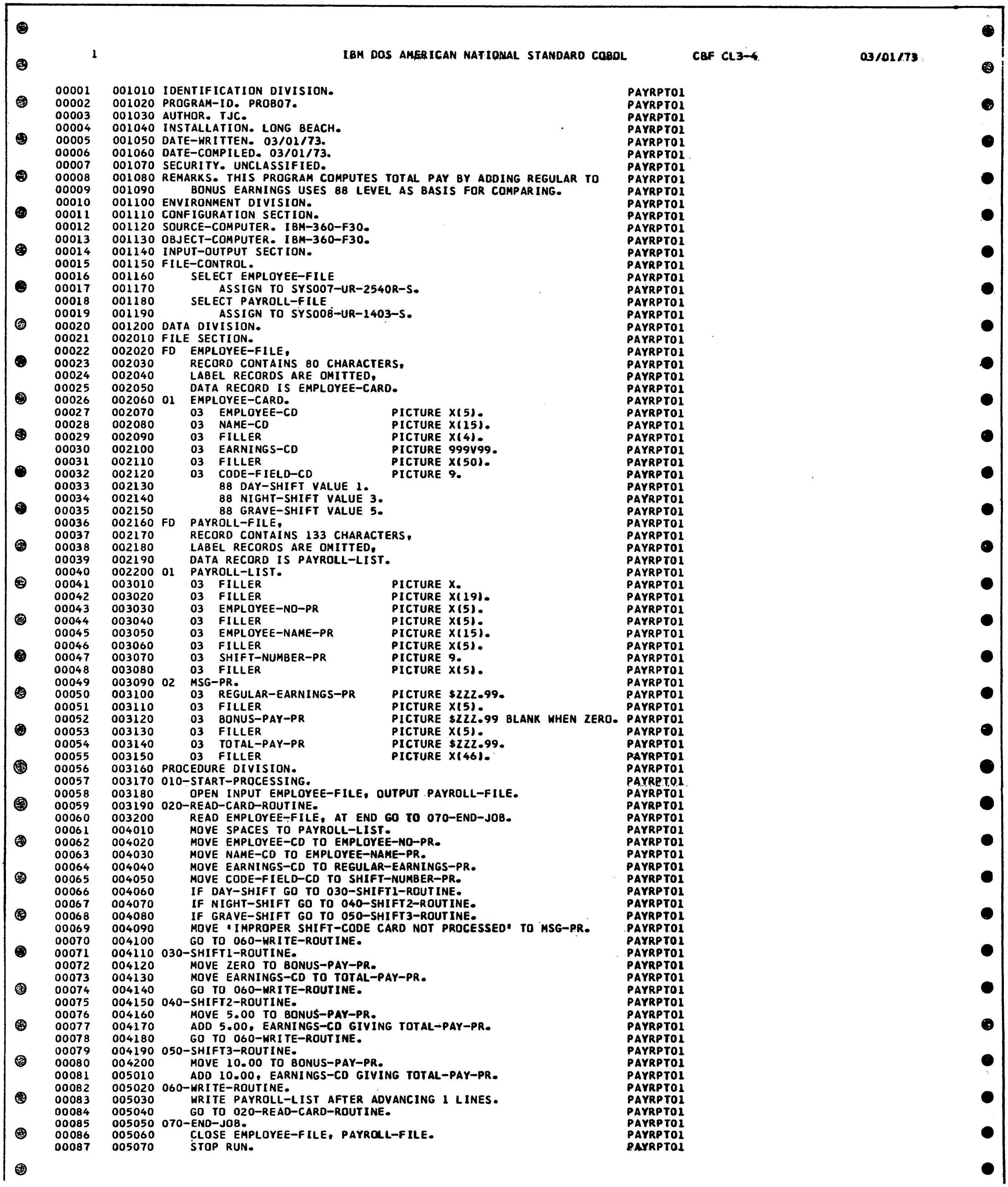

Flowcharting Worksheet
Programmer: TJC
Program No.: COBOL 07
Date:
Page: 1 of 1
Chart ID: C-07
Chart Name: BONUS PAY
Program Name: PAYROLL 03
START
DEFINE CARD AND PRINT FILES
010-START-PROCESSING
OPEN FILES
From C5
020-READ-CARD-ROUTINE
READ CARD
LAST CARD ?
YES
NO
CLEAR PRINT AREA
MOVE EMPLOYEE #, NAME, EARNINGS, AND CODE TO PRINT AREA
DAY-SHIFT (CODE 1)
NIGHT-SHIFT (CODE 3)
GRAVE-SHIFT (CODE 5)
From K1
MOVE ERROR MESSAGE TO PRINT AREA
From H1
030-SHIFT1-ROUTINE
MOVE ZEROS TO BONUS PAY
MOVE EARNINGS TO TOTAL-PAY
From J1
040-SHIFT2-ROUTINE
MOVE $5.00 TO BONUS PAY
ADD $5.00 AND EARNINGS GIVING TOTAL PAY
050-SHIFT3-ROUTINE
MOVE $10.00 TO BONUS PAY
ADD $10.00 AND EARNINGS GIVING TOTAL PAY
From A2, B3, B4
060-WRITE-ROUTINE
WRITE A LINE
From E1
070-END-JOB
CLOSE FILES
STOP RUN
A1
A2
A3
A4
A5
B1
B2
B3
B4
B5
C1
C2
C3
C4
C5
D1
D2
D3
D4
D5
E1
E2
E3
E4
E5
F1
F2
F3
F4
F5
G1
G2
G3
G4
G5
H1
H2
H3
H4
H5
J1
J2
J3
J4
J5
K1
K2
K3
K4
K5

Figure 7-7 Flowchart

Program

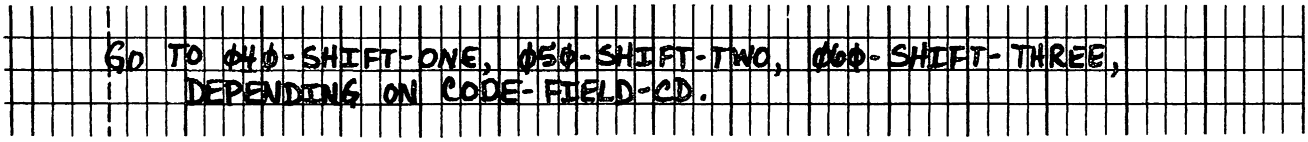


```
     1                                  IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4              03/01/73

00001   001010 IDENTIFICATION DIVISION.                                         PAYRPT01
00002   001020 PROGRAM-ID. PROB07.                                              PAYRPT01
00003   001030 AUTHOR. TJC.                                                     PAYRPT01
00004   001040 INSTALLATION. LONG BEACH.                                        PAYRPT01
00005   001050 DATE-WRITTEN. 03/01/73.                                          PAYRPT01
00006   001060 DATE-COMPILED. 03/01/73.                                         PAYRPT01
00007   001070 SECURITY. UNCLASSIFIED.                                          PAYRPT01
00008   001080 REMARKS. THIS PROGRAM COMPUTES TOTAL PAY BY ADDING REGULAR TO    PAYRPT01
00009   001090     BONUS EARNINGS USES 88 LEVEL AS BASIS FOR COMPARING.         PAYRPT01
00010   001100 ENVIRONMENT DIVISION.                                            PAYRPT01
00011   001110 CONFIGURATION SECTION.                                           PAYRPT01
00012   001120 SOURCE-COMPUTER. IBM-360-F30.                                    PAYRPT01
00013   001130 OBJECT-COMPUTER. IBM-360-F30.                                    PAYRPT01
00014   001140 INPUT-OUTPUT SECTION.                                            PAYRPT01
00015   001150 FILE-CONTROL.                                                    PAYRPT01
00016   001160     SELECT EMPLOYEE-FILE                                         PAYRPT01
00017   001170         ASSIGN TO SYS007-UR-2540R-S.                             PAYRPT01
00018   001180     SELECT PAYROLL-FILE                                          PAYRPT01
00019   001190         ASSIGN TO SYS008-UR-1403-S.                              PAYRPT01
00020   001200 DATA DIVISION.                                                   PAYRPT01
00021   002010 FILE SECTION.                                                    PAYRPT01
00022   002020 FD  EMPLOYEE-FILE,                                               PAYRPT01
00023   002030     RECORD CONTAINS 80 CHARACTERS,                               PAYRPT01
00024   002040     LABEL RECORDS ARE OMITTED,                                   PAYRPT01
00025   002050     DATA RECORD IS EMPLOYEE-CARD.                                PAYRPT01
00026   002060 01  EMPLOYEE-CARD.                                               PAYRPT01
00027   002070     03  EMPLOYEE-CD             PICTURE X(5).                    PAYRPT01
00028   002080     03  NAME-CD                 PICTURE X(15).                   PAYRPT01
00029   002090     03  FILLER                  PICTURE X(4).                    PAYRPT01
00030   002100     03  EARNINGS-CD             PICTURE 999V99.                  PAYRPT01
00031   002110     03  FILLER                  PICTURE X(50).                   PAYRPT01
00032   002120     03  CODE-FIELD-CD           PICTURE 9.                       PAYRPT01
00033   002130         88 DAY-SHIFT VALUE 1.                                    PAYRPT01
00034   002140         88 NIGHT-SHIFT VALUE 3.                                  PAYRPT01
00035   002150         88 GRAVE-SHIFT VALUE 5.                                  PAYRPT01
00036   002160 FD  PAYROLL-FILE,                                                PAYRPT01
00037   002170     RECORD CONTAINS 133 CHARACTERS,                              PAYRPT01
00038   002180     LABEL RECORDS ARE OMITTED,                                   PAYRPT01
00039   002190     DATA RECORD IS PAYROLL-LIST.                                 PAYRPT01
00040   002200 01  PAYROLL-LIST.                                                PAYRPT01
00041   003010     03  FILLER                  PICTURE X.                       PAYRPT01
00042   003020     03  FILLER                  PICTURE X(19).                   PAYRPT01
00043   003030     03  EMPLOYEE-NO-PR          PICTURE X(5).                    PAYRPT01
00044   003040     03  FILLER                  PICTURE X(5).                    PAYRPT01
00045   003050     03  EMPLOYEE-NAME-PR        PICTURE X(15).                   PAYRPT01
00046   003060     03  FILLER                  PICTURE X(5).                    PAYRPT01
00047   003070     03  SHIFT-NUMBER-PR         PICTURE 9.                       PAYRPT01
00048   003080     03  FILLER                  PICTURE X(5).                    PAYRPT01
00049   003090 02  MSG-PR.                                                      PAYRPT01
00050   003100     03  REGULAR-EARNINGS-PR     PICTURE $ZZZ.99.                 PAYRPT01
00051   003110     03  FILLER                  PICTURE X(5).                    PAYRPT01
00052   003120     03  BONUS-PAY-PR            PICTURE $ZZZ.99 BLANK WHEN ZERO. PAYRPT01
00053   003130     03  FILLER                  PICTURE X(5).                    PAYRPT01
00054   003140     03  TOTAL-PAY-PR            PICTURE $ZZZ.99.                 PAYRPT01
00055   003150     03  FILLER                  PICTURE X(46).                   PAYRPT01
00056   003160 PROCEDURE DIVISION.                                              PAYRPT01
00057   003170 010-START-PROCESSING.                                            PAYRPT01
00058   003180     OPEN INPUT EMPLOYEE-FILE, OUTPUT PAYROLL-FILE.               PAYRPT01
00059   003190 020-READ-CARD-ROUTINE.                                           PAYRPT01
00060   003200     READ EMPLOYEE-FILE, AT END GO TO 070-END-JOB.                PAYRPT01
00061   004010     MOVE SPACES TO PAYROLL-LIST.                                 PAYRPT01
00062   004020     MOVE EMPLOYEE-CD TO EMPLOYEE-NO-PR.                          PAYRPT01
00063   004030     MOVE NAME-CD TO EMPLOYEE-NAME-PR.                            PAYRPT01
00064   004040     MOVE EARNINGS-CD TO REGULAR-EARNINGS-PR.                     PAYRPT01
00065   004050     MOVE CODE-FIELD-CD TO SHIFT-NUMBER-PR.                       PAYRPT01
00066   004060     IF DAY-SHIFT GO TO 030-SHIFT1-ROUTINE.                       PAYRPT01
00067   004070     IF NIGHT-SHIFT GO TO 040-SHIFT2-ROUTINE.                     PAYRPT01
00068   004080     IF GRAVE-SHIFT GO TO 050-SHIFT3-ROUTINE.                     PAYRPT01
00069   004090     MOVE 'IMPROPER SHIFT-CODE CARD NOT PROCESSED' TO MSG-PR.     PAYRPT01
00070   004100     GO TO 060-WRITE-ROUTINE.                                     PAYRPT01
00071   004110 030-SHIFT1-ROUTINE.                                              PAYRPT01
00072   004120     MOVE ZERO TO BONUS-PAY-PR.                                   PAYRPT01
00073   004130     MOVE EARNINGS-CD TO TOTAL-PAY-PR.                            PAYRPT01
00074   004140     GO TO 060-WRITE-ROUTINE.                                     PAYRPT01
00075   004150 040-SHIFT2-ROUTINE.                                              PAYRPT01
00076   004160     MOVE 5.00 TO BONUS-PAY-PR.                                   PAYRPT01
00077   004170     ADD 5.00, EARNINGS-CD GIVING TOTAL-PAY-PR.                   PAYRPT01
00078   004180     GO TO 060-WRITE-ROUTINE.                                     PAYRPT01
00079   004190 050-SHIFT3-ROUTINE.                                              PAYRPT01
00080   004200     MOVE 10.00 TO BONUS-PAY-PR.                                  PAYRPT01
00081   005010     ADD 10.00, EARNINGS-CD GIVING TOTAL-PAY-PR.                  PAYRPT01
00082   005020 060-WRITE-ROUTINE.                                               PAYRPT01
00083   005030     WRITE PAYROLL-LIST AFTER ADVANCING 1 LINES.                  PAYRPT01
00084   005040     GO TO 020-READ-CARD-ROUTINE.                                 PAYRPT01
00085   005050 070-END-JOB.                                                     PAYRPT01
00086   005060     CLOSE EMPLOYEE-FILE, PAYROLL-FILE.                           PAYRPT01
00087   005070     STOP RUN.                                                    PAYRPT01
```

Figure 7-8 COBOL Program

GO TO . . . . DEPENDING

Another technique useful when it is necessary to execute various routines depending upon the presence of a particular type of control punch on a card is the use of Option 2 of the GO TO Statement. A sample of the use of this Statement in the Procedure Division is illustrated below.



```
GO TO 040-SHIFT-ONE, 050-SHIFT-TWO, 060-SHIFT-THREE,
    DEPENDING ON CODE-FIELD-CD.
```

Figure 7-9 Example of GO TO Statement

The format notation is as follows:

```
GO TO procedure-name-1 [procedure-name-2] ...
   DEPENDING ON identifier
```

Figure 7-10 Format Notation – GO TO Statement

In this GO TO statement control goes to the 1st, 2nd, . . . , nth procedure-name as the value of identifier is 1, 2, . . . , n. If identifier has a value outside the range 1 to n, no transfer takes place, and control passes to the next statement after the GO TO statement.

Identifier must be an elementary integral numeric item whose length does not exceed four digits and whose usage is either DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3.

Thus, in the sample GO TO statement illustrated in Figure 7-9, if CODE-FIELD-CD takes on the value of "1", the 040-SHIFT-ONE Routine is executed; if CODE-FIELD-CD takes on the value of "2", the 050-SHIFT-TWO Routine is executed, and if CODE-FIELD-CD takes on the value of "3", then the 060-SHIFT-THREE Routine is executed.

Because only three procedure names are specified (040-SHIFT-ONE, 050-SHIFT-TWO, and 060-SHIFT-THREE), the value of CODE-FIELD-CD can only be "1", "2", or "3". If the CODE-FIELD-CD contains a "4" punch (an error), the next instruction in sequence would be executed.

INPUT EDITING

In business applications the data to be processed should be "validated" to assure that each record contains the proper control punch, that fields involved in calculations are numeric, that fields involved in calculations do not contain blanks, etc. This type of checking may be performed by means of a separate "input editing" program, or by the use of special checking routines included in the program that will process the data.

The sample program contained on page 7.16 illustrates a form of "input editing". In this problem input consists of punched cards containing an Employee Number Field, an Employee Name Field, a Regular Earnings Field, an Overtime Earnings Field, and a Code Field containing a "1", "2", or "3" control punch. If the input cards DO NOT contain a "1", "2", or "3" punch, an error message is to be printed on the report.

When checking for proper control punches the programmer should assure that all possible conditions are covered. For example, when using the "GO TO . . . DEPENDING" Option, if the field containing the control punch is blank, a "data exception" will occur and processing will be terminated. If the control punch field contains a letter of the alphabet "A" (12-1 punch on the card), the program would execute the routine related to a digit "1" punch in the Code Field.

The following is a segment of a program which illustrates a suggested technique for checking for valid control punches on the card.

```
IF BONUS-CODE IS ALPHABETIC OR BONUS-CODE IS NOT NUMERIC
    GO TO 030-ERROR.
```

Figure 7-11 Example of Class Test

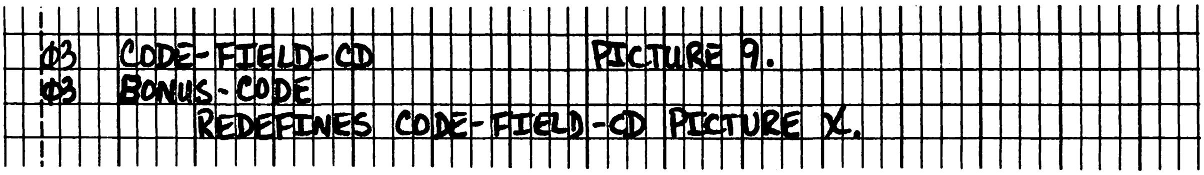
To check for all possible conditions including the presence of letters of the alphabet or special characters in the Code Field, the Statement IF BONUS-CODE IS ALPHABETIC OR BONUS-CODE IS NOT NUMERIC GO TO 030-ERROR is included in the program.

When using the GO TO . . . . DEPENDING Statement, the field referenced following the word "DEPENDING" (CODE-FIELD) must be numeric, that is, defined by a PICTURE containing 9's; however, the Statement IF ALPHABETIC is not valid when referencing a field defined as numeric (Picture 9).

REDEFINES CLAUSE

To permit an Alphabetic Class Test for a single digit field used in a GO TO . . . . DEPENDING Statement it is necessary to define the field as numeric and to REDEFINE the field as alphanumeric (Picture X).

The following is a segment of the Data Division in which a Field with the name CODE-FIELD-CD is defined as numeric and REDEFINED as alphanumeric with the name BONUS-CODE and with a Picture X.



```
03  CODE-FIELD-CD               PICTURE 9.
03  BONUS-CODE
        REDEFINES CODE-FIELD-CD PICTURE X.
```

Figure 7-12 Example of Redefines

The REDEFINES Clause specifies that the same area is to contain different data items, or provides an alternative grouping or description of data.

The format of the REDEFINES Clause is illustrated below.

```
data-name-1 [REDEFINES data-name-2]
```

Figure 7-13 Format Notation – REDEFINES Clause

When written, the REDEFINES clause must be the first clause following the Data-Name that defines the entry. Thus, from Figure 7-12 it can be seen that the REDEFINES clause immediately follows the data name BONUS-CODE. It must be noted that the Redefines clause merely gives a different symbolic name or a different data format to the same area of main storage. It in no way affects what data is to be stored in the area – it merely informs the compiler how the data within the area of main storage is to be treated.

For example, the following diagram illustrates the value Zero (X'F0') in the area referenced by CODE-FIELD-CD and BONUS-CODE.

EXAMPLE

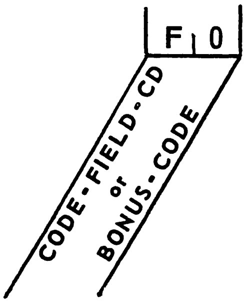


Figure 7-14 Example of Result of Redefines Clause

Note in the example in Figure 7-14 that the same character in the same position of main storage is given two different names, CODE-FIELD-CD and BONUS-CD. In addition, from the example illustrated in Figure 7-12, it can be seen these data-names have different picture attributes. The field CODE-FIELD-CD is defined as a Display Numeric field (Picture 9) and the BONUS-CODE field is defined as an alphameric field (Picture X). The compiler will treat these fields differently even though they reference the same core storage location.

Defining the CODE-FIELD-CD field as numeric allows the field to be referenced in the GO TO . . . DEPENDING statement and redefining the field as alphameric allows the field to be tested for the presence of alphabetic characters.

It should again be noted that the statement IF ALPHABETIC is an invalid statement for any field defined as numeric. Therefore, if a field defined as numeric is to be tested as "alphabetic", it must be redefined.

If a field is defined as alphanumeric (Picture X) and the last character of a field containing numbers contains a digit with a sign overpunch, the field is considered NUMERIC. For a single character alphanumeric field containing a digit with a sign overpunch, the tests IF NUMERIC AND IF ALPHABETIC will both be considered true. Thus, in the sample Class Test illustrated in Figure 7-11, the statement IF ALPHABETIC will detect control punches containing letters of the alphabet and the statement "NOT NUMERIC" would detect special characters.

EXAMINE CLAUSE

In the sample problem illustrated on subsequent pages, Total Earnings are to be calculated by adding the Regular Earnings field, Overtime field, and Bonus-Pay. If there is no overtime, this amount field will be blank on the input data card. It should be noted that a blank field cannot be involved in a calculation.

One technique to handle problems of this type is to use the EXAMINE Statement in the Procedure Division. The EXAMINE Statement may be used to replace blanks in a field with zeros. The following segments of a COBOL program illustrate the entries in the Data Division for the input data card, and the entry in the Procedure Division using the EXAMINE Statement.

```
01  TIME-CARD.
    03  EMPLOYEE-NO-CD                  PICTURE 9(4).
    03  EMPLOYEE-NAME-CD                PICTURE X(20).
    03  REGULAR-EARNINGS-CD             PICTURE 999V99.
    03  OVER-TIME-CD                    PICTURE 999V99.
    03  EXTRA-HOURS
            REDEFINES OVER-TIME-CD PICTURE X(5).
    03  FILLER                          PICTURE X(45).
    03  CODE-FIELD-CD                   PICTURE 9.
    03  BONUS-CODE
            REDEFINES CODE-FIELD-CD PICTURE X.
      -
PROCEDURE DIVISION.
      -
    EXAMINE EXTRA-HOURS REPLACING ALL ' ' BY '0'.
```

Figure 7-15 Sample Data Division with REDEFINES Clause and EXAMINE Statement

The effect of the EXAMINE Statement is to replace any blanks in the Overtime Field with zeros, so that the field may be used in calculations.

Note that the character being replaced is enclosed in single quotes indicating that this data item is a non-numeric literal. Because non-numeric literals are required in an EXAMINE Statement and numeric fields are required in arithmetic statements it is necessary to define the field referenced in the arithmetic statement as numeric (Picture 9's), and redefine the same field as alphanumeric so that the field may be referenced in the EXAMINE Statement.

The format notation of the EXAMINE Statement is illustrated below.

```
Format 1
                                     ( UNTIL FIRST )
  EXAMINE identifier TALLYING       {  ALL          }  literal-1
                                     ( LEADING     )

          [REPLACING BY literal-2]


Format 2
                                     ( ALL         )
                                     ) LEADING     (
  EXAMINE identifier REPLACING      {  FIRST        }  literal-1
                                     ( UNTIL FIRST )

          BY  literal-2
```

Figure 7-16 Format Notation – EXAMINE Statement

The EXAMINE Statement is used to replace certain occurrences of a given character and/or to count the number of such occurrences in a data item.

Identifier must refer to a data item whose USAGE is DISPLAY.

Literal-1 and literal-2 must be single-character non-numeric literals (i.e., enclosed in quotation marks) and members of the set of allowable characters for the data item. For example, a "2" cannot replace an "A" in an alphabetic item, but may do so in an alphanumeric item.

The use of figurative constants instead of "literal-1" is permitted.

SAMPLE PROBLEM

The following sample problem illustrates an application utilizing the REDEFINES Clause, the EXAMINE Statement, and the GO TO . . . . DEPENDING Statement. Input consist of Employee Earnings Cards, containing the Employee Number, the Employee Name, Regular Earnings, Overtime Earnings, and a Code Field. If there are no Overtime Earnings, this field is blank on the input data card.

Input

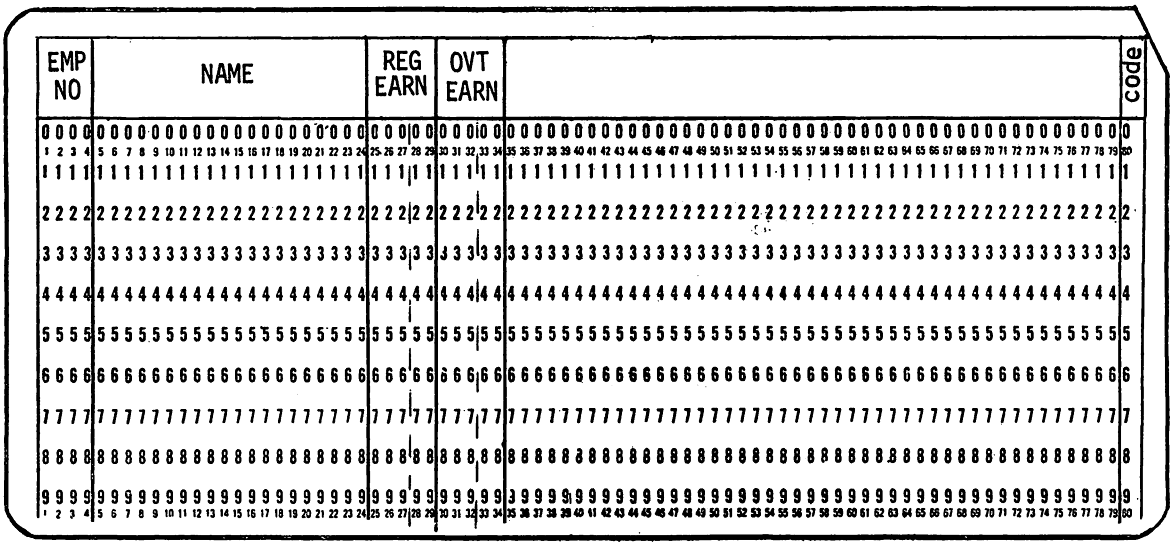


Figure 7-17 Input

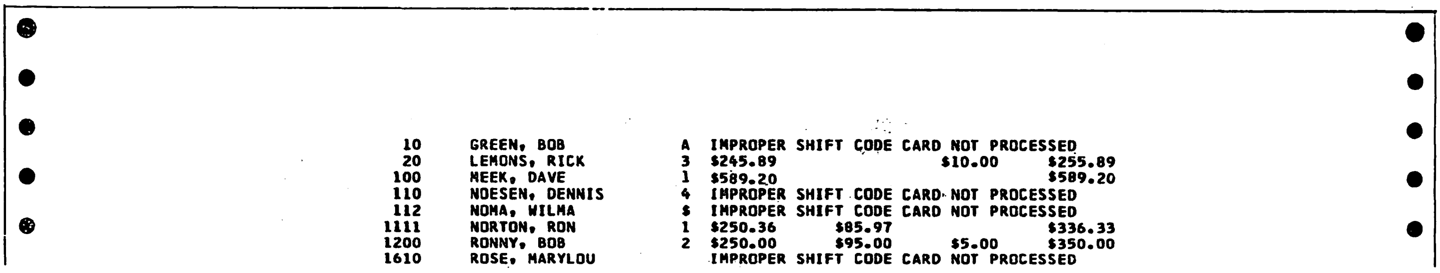
Output

Output is to consist of a printed report listing Total Earnings. Total Earnings are obtained by adding Regular Earnings, Overtime Earnings, plus Bonus Pay.

If there is a "1" punch in the Code Field of the Employee Earnings Card, it indicates that the employee works on Shift One and is to receive no "bonus pay".

If there is a "2" punch in the card, it indicates that the employee works on Shift Two and is to receive $5.00 in "bonus pay".

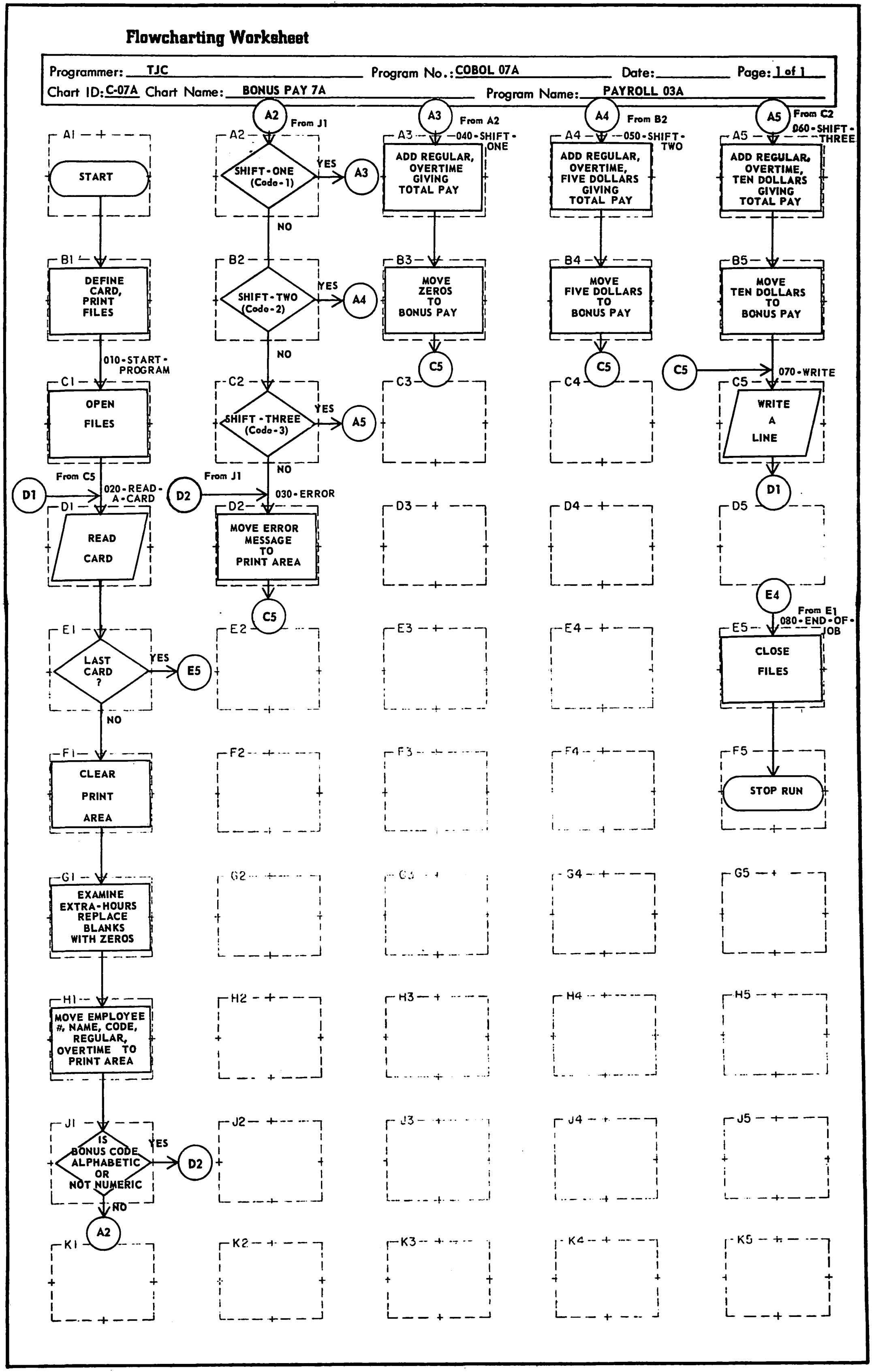
If there is a "3" punch in the card, it indicates that the employee works on Shift Three and is to receive $10.00 in "bonus pay".



```
  10     GREEN, BOB          A  IMPROPER SHIFT CODE CARD NOT PROCESSED
  20     LEMONS, RICK        3  $245.89                $10.00     $255.89
 100     MEEK, DAVE          1  $589.20                           $589.20
 110     NOESEN, DENNIS      4  IMPROPER SHIFT CODE CARD NOT PROCESSED
 112     NOMA, WILMA         $  IMPROPER SHIFT CODE CARD NOT PROCESSED
1111     NORTON, RON         1  $250.36     $85.97                $336.33
1200     RONNY, BOB          2  $250.00     $95.00      $5.00     $350.00
1610     ROSE, MARYLOU          IMPROPER SHIFT CODE CARD NOT PROCESSED
```

Figure 7-18 Output

Flowchart

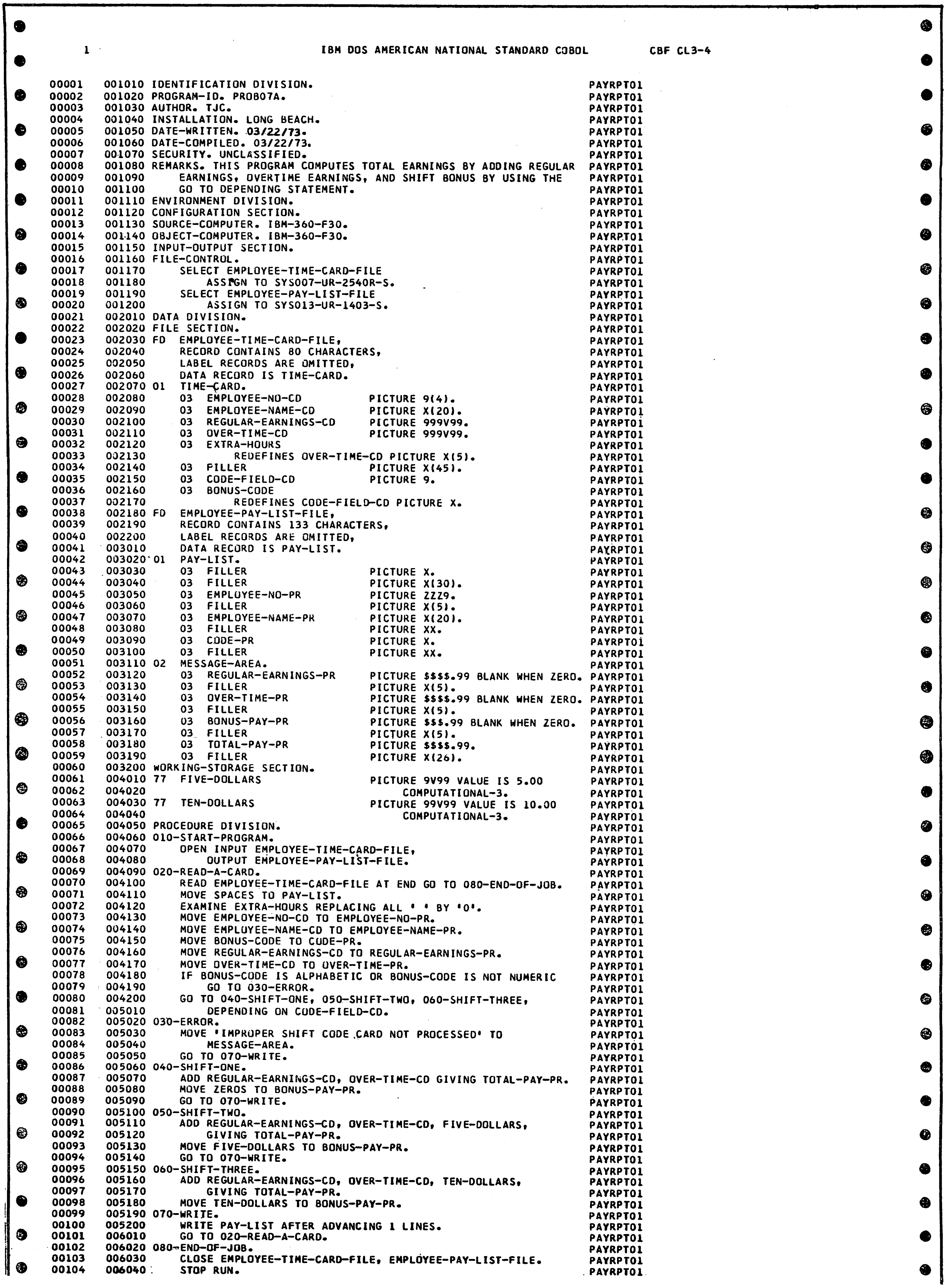


Flowcharting Worksheet

Programmer: TJC
Program No.: COBOL 07A
Date:
Page: 1 of 1
Chart ID: C-07A
Chart Name: BONUS PAY 7A
Program Name: PAYROLL 03A

A1
START
B1
DEFINE CARD, PRINT FILES
010-START-PROGRAM
C1
OPEN FILES
From C5
D1
020-READ-A-CARD
D1
READ CARD
E1
LAST CARD ?
YES
E5
NO
F1
CLEAR PRINT AREA
G1
EXAMINE EXTRA-HOURS REPLACE BLANKS WITH ZEROS
H1
MOVE EMPLOYEE #, NAME, CODE, REGULAR, OVERTIME TO PRINT AREA
J1
IS BONUS CODE ALPHABETIC OR NOT NUMERIC
YES
D2
NO
A2
K1
A2
From J1
A2
SHIFT-ONE (Code-1)
YES
A3
NO
B2
SHIFT-TWO (Code-2)
YES
A4
NO
C2
SHIFT-THREE (Code-3)
YES
A5
NO
From J1
D2
030-ERROR
D2
MOVE ERROR MESSAGE TO PRINT AREA
C5
E2
F2
G2
H2
J2
K2
A3
From A2
A3
040-SHIFT-ONE
ADD REGULAR, OVERTIME GIVING TOTAL PAY
B3
MOVE ZEROS TO BONUS PAY
C5
C3
D3
E3
F3
G3
H3
J3
K3
A4
From B2
A4
050-SHIFT-TWO
ADD REGULAR, OVERTIME, FIVE DOLLARS GIVING TOTAL PAY
B4
MOVE FIVE DOLLARS TO BONUS PAY
C5
C4
D4
E4
F4
G4
H4
J4
K4
A5
From C2
060-SHIFT-THREE
A5
ADD REGULAR, OVERTIME, TEN DOLLARS GIVING TOTAL PAY
B5
MOVE TEN DOLLARS TO BONUS PAY
C5
070-WRITE
C5
WRITE A LINE
D1
D5
E4
From E1
080-END-OF-JOB
E5
CLOSE FILES
F5
STOP RUN
G5
H5
J5
K5

Figure 7-19 Flowchart



```
     1                                  IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4

00001   001010 IDENTIFICATION DIVISION.                                         PAYRPT01
00002   001020 PROGRAM-ID. PROB07A.                                             PAYRPT01
00003   001030 AUTHOR. TJC.                                                     PAYRPT01
00004   001040 INSTALLATION. LONG BEACH.                                        PAYRPT01
00005   001050 DATE-WRITTEN. 03/22/73.                                          PAYRPT01
00006   001060 DATE-COMPILED. 03/22/73.                                         PAYRPT01
00007   001070 SECURITY. UNCLASSIFIED.                                          PAYRPT01
00008   001080 REMARKS. THIS PROGRAM COMPUTES TOTAL EARNINGS BY ADDING REGULAR  PAYRPT01
00009   001090     EARNINGS, OVERTIME EARNINGS, AND SHIFT BONUS BY USING THE    PAYRPT01
00010   001100     GO TO DEPENDING STATEMENT.                                   PAYRPT01
00011   001110 ENVIRONMENT DIVISION.                                            PAYRPT01
00012   001120 CONFIGURATION SECTION.                                           PAYRPT01
00013   001130 SOURCE-COMPUTER. IBM-360-F30.                                    PAYRPT01
00014   001140 OBJECT-COMPUTER. IBM-360-F30.                                    PAYRPT01
00015   001150 INPUT-OUTPUT SECTION.                                            PAYRPT01
00016   001160 FILE-CONTROL.                                                    PAYRPT01
00017   001170     SELECT EMPLOYEE-TIME-CARD-FILE                               PAYRPT01
00018   001180         ASSIGN TO SYS007-UR-2540R-S.                             PAYRPT01
00019   001190     SELECT EMPLOYEE-PAY-LIST-FILE                                PAYRPT01
00020   001200         ASSIGN TO SYS013-UR-1403-S.                              PAYRPT01
00021   002010 DATA DIVISION.                                                   PAYRPT01
00022   002020 FILE SECTION.                                                    PAYRPT01
00023   002030 FD  EMPLOYEE-TIME-CARD-FILE,                                     PAYRPT01
00024   002040     RECORD CONTAINS 80 CHARACTERS,                               PAYRPT01
00025   002050     LABEL RECORDS ARE OMITTED,                                   PAYRPT01
00026   002060     DATA RECORD IS TIME-CARD.                                    PAYRPT01
00027   002070 01  TIME-CARD.                                                   PAYRPT01
00028   002080     03  EMPLOYEE-NO-CD          PICTURE 9(4).                    PAYRPT01
00029   002090     03  EMPLOYEE-NAME-CD        PICTURE X(20).                   PAYRPT01
00030   002100     03  REGULAR-EARNINGS-CD     PICTURE 999V99.                  PAYRPT01
00031   002110     03  OVER-TIME-CD            PICTURE 999V99.                  PAYRPT01
00032   002120     03  EXTRA-HOURS                                              PAYRPT01
00033   002130             REDEFINES OVER-TIME-CD PICTURE X(5).                 PAYRPT01
00034   002140     03  FILLER                  PICTURE X(45).                   PAYRPT01
00035   002150     03  CODE-FIELD-CD           PICTURE 9.                       PAYRPT01
00036   002160     03  BONUS-CODE                                               PAYRPT01
00037   002170             REDEFINES CODE-FIELD-CD PICTURE X.                   PAYRPT01
00038   002180 FD  EMPLOYEE-PAY-LIST-FILE,                                      PAYRPT01
00039   002190     RECORD CONTAINS 133 CHARACTERS,                              PAYRPT01
00040   002200     LABEL RECORDS ARE OMITTED,                                   PAYRPT01
00041   003010     DATA RECORD IS PAY-LIST.                                     PAYRPT01
00042   003020 01  PAY-LIST.                                                    PAYRPT01
00043   003030     03  FILLER                  PICTURE X.                       PAYRPT01
00044   003040     03  FILLER                  PICTURE X(30).                   PAYRPT01
00045   003050     03  EMPLOYEE-NO-PR          PICTURE ZZZ9.                    PAYRPT01
00046   003060     03  FILLER                  PICTURE X(5).                    PAYRPT01
00047   003070     03  EMPLOYEE-NAME-PR        PICTURE X(20).                   PAYRPT01
00048   003080     03  FILLER                  PICTURE XX.                      PAYRPT01
00049   003090     03  CODE-PR                 PICTURE X.                       PAYRPT01
00050   003100     03  FILLER                  PICTURE XX.                      PAYRPT01
00051   003110 02  MESSAGE-AREA.                                                PAYRPT01
00052   003120     03  REGULAR-EARNINGS-PR     PICTURE $$$$.99 BLANK WHEN ZERO. PAYRPT01
00053   003130     03  FILLER                  PICTURE X(5).                    PAYRPT01
00054   003140     03  OVER-TIME-PR            PICTURE $$$$.99 BLANK WHEN ZERO. PAYRPT01
00055   003150     03  FILLER                  PICTURE X(5).                    PAYRPT01
00056   003160     03  BONUS-PAY-PR            PICTURE $$$.99 BLANK WHEN ZERO.  PAYRPT01
00057   003170     03  FILLER                  PICTURE X(5).                    PAYRPT01
00058   003180     03  TOTAL-PAY-PR            PICTURE $$$$.99.                 PAYRPT01
00059   003190     03  FILLER                  PICTURE X(26).                   PAYRPT01
00060   003200 WORKING-STORAGE SECTION.                                         PAYRPT01
00061   004010 77  FIVE-DOLLARS                PICTURE 9V99 VALUE IS 5.00       PAYRPT01
00062   004020                                     COMPUTATIONAL-3.             PAYRPT01
00063   004030 77  TEN-DOLLARS                 PICTURE 99V99 VALUE IS 10.00     PAYRPT01
00064   004040                                     COMPUTATIONAL-3.             PAYRPT01
00065   004050 PROCEDURE DIVISION.                                              PAYRPT01
00066   004060 010-START-PROGRAM.                                               PAYRPT01
00067   004070     OPEN INPUT EMPLOYEE-TIME-CARD-FILE,                          PAYRPT01
00068   004080         OUTPUT EMPLOYEE-PAY-LIST-FILE.                           PAYRPT01
00069   004090 020-READ-A-CARD.                                                 PAYRPT01
00070   004100     READ EMPLOYEE-TIME-CARD-FILE AT END GO TO 080-END-OF-JOB.    PAYRPT01
00071   004110     MOVE SPACES TO PAY-LIST.                                     PAYRPT01
00072   004120     EXAMINE EXTRA-HOURS REPLACING ALL ' ' BY '0'.                PAYRPT01
00073   004130     MOVE EMPLOYEE-NO-CD TO EMPLOYEE-NO-PR.                       PAYRPT01
00074   004140     MOVE EMPLOYEE-NAME-CD TO EMPLOYEE-NAME-PR.                   PAYRPT01
00075   004150     MOVE BONUS-CODE TO CODE-PR.                                  PAYRPT01
00076   004160     MOVE REGULAR-EARNINGS-CD TO REGULAR-EARNINGS-PR.             PAYRPT01
00077   004170     MOVE OVER-TIME-CD TO OVER-TIME-PR.                           PAYRPT01
00078   004180     IF BONUS-CODE IS ALPHABETIC OR BONUS-CODE IS NOT NUMERIC     PAYRPT01
00079   004190         GO TO 030-ERROR.                                         PAYRPT01
00080   004200     GO TO 040-SHIFT-ONE, 050-SHIFT-TWO, 060-SHIFT-THREE,         PAYRPT01
00081   005010         DEPENDING ON CODE-FIELD-CD.                              PAYRPT01
00082   005020 030-ERROR.                                                       PAYRPT01
00083   005030     MOVE 'IMPROPER SHIFT CODE CARD NOT PROCESSED' TO             PAYRPT01
00084   005040         MESSAGE-AREA.                                            PAYRPT01
00085   005050     GO TO 070-WRITE.                                             PAYRPT01
00086   005060 040-SHIFT-ONE.                                                   PAYRPT01
00087   005070     ADD REGULAR-EARNINGS-CD, OVER-TIME-CD GIVING TOTAL-PAY-PR.   PAYRPT01
00088   005080     MOVE ZEROS TO BONUS-PAY-PR.                                  PAYRPT01
00089   005090     GO TO 070-WRITE.                                             PAYRPT01
00090   005100 050-SHIFT-TWO.                                                   PAYRPT01
00091   005110     ADD REGULAR-EARNINGS-CD, OVER-TIME-CD, FIVE-DOLLARS,         PAYRPT01
00092   005120         GIVING TOTAL-PAY-PR.                                     PAYRPT01
00093   005130     MOVE FIVE-DOLLARS TO BONUS-PAY-PR.                           PAYRPT01
00094   005140     GO TO 070-WRITE.                                             PAYRPT01
00095   005150 060-SHIFT-THREE.                                                 PAYRPT01
00096   005160     ADD REGULAR-EARNINGS-CD, OVER-TIME-CD, TEN-DOLLARS,          PAYRPT01
00097   005170         GIVING TOTAL-PAY-PR.                                     PAYRPT01
00098   005180     MOVE TEN-DOLLARS TO BONUS-PAY-PR.                            PAYRPT01
00099   005190 070-WRITE.                                                       PAYRPT01
00100   005200     WRITE PAY-LIST AFTER ADVANCING 1 LINES.                      PAYRPT01
00101   006010     GO TO 020-READ-A-CARD.                                       PAYRPT01
00102   006020 080-END-OF-JOB.                                                  PAYRPT01
00103   006030     CLOSE EMPLOYEE-TIME-CARD-FILE, EMPLOYEE-PAY-LIST-FILE.       PAYRPT01
00104   006040     STOP RUN.                                                    PAYRPT01
```

Figure 7-20 COBOL Program

PROGRAMMING ASSIGNMENT 1

CONDITION NAMES

INSTRUCTIONS

From the sales cards illustrated below, a Current Sales Quota Report is to be prepared. Sales Quotas have been established as follows:

EMPLOYEE CLASSIFICATION	CURRENT SALES QUOTA
MANAGER	$1,000.00
FULL TIME	$ 750.00
PART TIME	$ 500.00

Managers are identified by a "1" punch in the Code field, card column 80. Full time Employee cards contain a "2" punch in the Code field, and Part time Employee cards contain a "3" punch in the Code field.

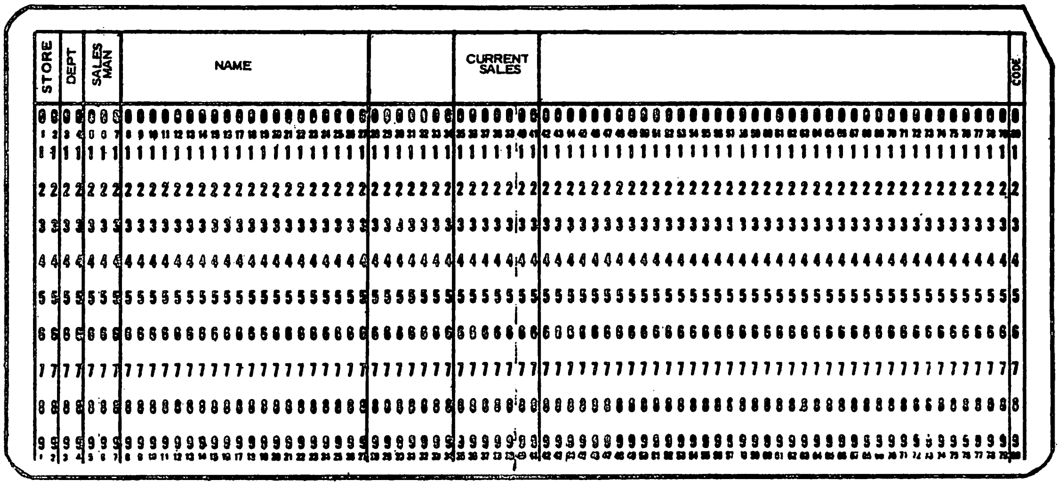
If Current Sales are below quota, print the identifying message "BELOW QUOTA" on the report. Managers are to be identified by printing the word "MANAGER" on the report.

If the cards contain an invalid punch in the Code field, no comparison is to be made and the message "INVALID EMPLOYEE CODE" is to be printed.

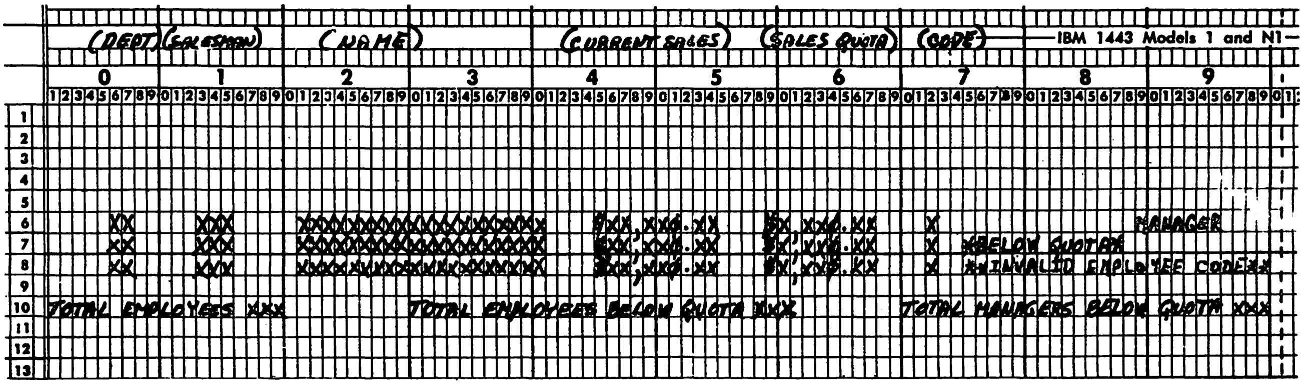
Cards are to be sorted by Salesman Number and Department Number. Test data is contained in the appendices.

Incorporate CONDITION Names in the COBOL program developed to prepare the report.

INPUT: Sales Cards



OUTPUT: Current Sales Quota Report


IBM 1443 Models 1 and N1

(DEPT) (SALESMAN) (NAME) (CURRENT SALES) (SALES QUOTA) (CODE)

```
     XX     XXX     XXXXXXXXXXXXXXXXXXXXXX   $XX,XX0.XX    $X,XX0.XX    X                    MANAGER
     XX     XXX     XXXXXXXXXXXXXXXXXXXXXX   $XX,XX0.XX    $X,XX0.XX    X   *BELOW QUOTA*
     XX     XXX     XXXXXXXXXXXXXXXXXXXXXX   $XX,XX0.XX    $X,XX0.XX    X   **INVALID EMPLOYEE CODE**

TOTAL EMPLOYEES XXX           TOTAL EMPLOYEES BELOW QUOTA XXX         TOTAL MANAGERS BELOW QUOTA XXX
```

CHAPTER 7

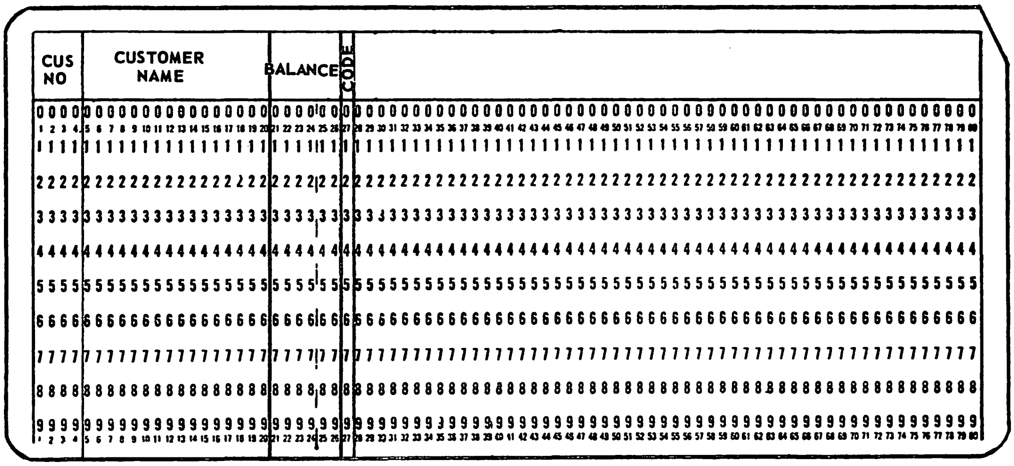
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

Write the COBOL program required to produce a Customer Sales Report. Input cards will contain a control code indicating whether the customer balance punched on the card is current, whether the balance is over 30 days old, or whether the balance is over 60 days old. A "1" punch indicates a current account, a "2" punch indicates an account 31-60 days past due; and a "3" punch indicates an account over 60 days past due. If the account is current, a 3% cash discount is given. If the account is 31-60 days past due, no cash discount is given. For accounts over 60 days past due, a 1.5% service charge is added to the account.

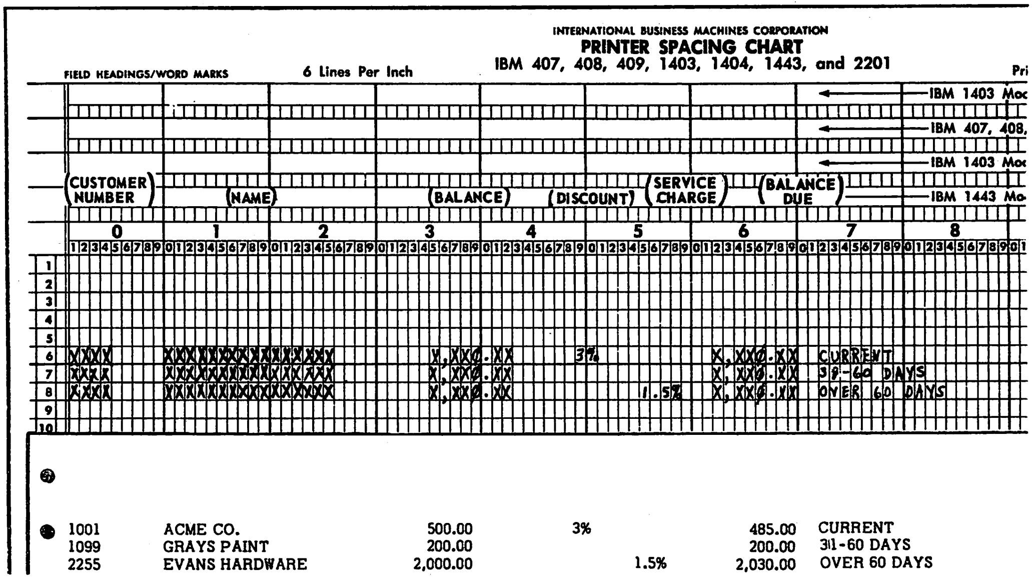
INPUT: Customer Sales Cards

Input is to consist of Customer Sales cards containing the Customer Number, the Customer Name, the Balance and a Code field containing a "1", "2", or "3" punch. The format of the Sales cards is illustrated below.



OUTPUT: Customer Sales Report

Output is to consist of a Customer Sales Report. The report is to contain the Customer Number, the Customer Name, the Balance, the Discount Percentage or Service Charge, and the Balance Due. The Balance Due is obtained by multiplying the Balance by the Discount Percentage and subtracting the answer obtained from the original balance, or by multiplying the balance by the Service Charge and adding the answer to the original balance. Each Customer is to be identified as being "Current", "31-60 Days" or "Over 60 Days".


INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, and 2201
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
IBM 1403 Mo
IBM 407, 408,
IBM 1403 Mo
IBM 1443 Mo
CUSTOMER NUMBER
NAME
BALANCE
DISCOUNT
SERVICE CHARGE
BALANCE DUE
XXXX
XXXXXXXXXXXXXXXX
X,XXØ.XX
3%
1.5%
CURRENT
31-60 DAYS
OVER 60 DAYS

```
1001      ACME CO.                 500.00       3%                 485.00  CURRENT
1099      GRAYS PAINT              200.00                          200.00  31-60 DAYS
2255      EVANS HARDWARE         2,000.00              1.5%      2,030.00  OVER 60 DAYS
```

TEST DATA

CUS NO (col 1-4)	NAME (col 5-20)	BALANCE (col 21-26)	CODE (col 27)
1001	ACME CO.	050000	1
1099	GRAYS PAINT	020000	2
2255	EVANS HARDWARE	200000	3
3711	FRANKEL SHOES	129371	2
4573	JOHNSON NURSERY	070050	1
7243	STRONG BARBELLS	027543	3
8973	ZEITH PENS	000043	3

CHAPTER 8

HEADINGS – PRINT OVERFLOW

INTRODUCTION

The programmer has great flexibility when programming with COBOL in controlling such functions as stacker selection, single, double, and triple spacing on a report, the printing of column and report headings, and the skipping to a new page on a report after a predetermined number of lines have been printed on the previous form.

The operations of single, double, and triple spacing on the printer are controlled by use of the AFTER ADVANCING Option.

When spacing greater than single, double, or triple spacing is to occur, a special carriage control tape is normally used. This tape is positioned in the printer, and when used in conjunction with related COBOL Statements in the program offers great flexibility in controlling the spacing on a form.

The following diagram of the carriage control tape illustrates the relationship of the carriage control tape to the printed report.

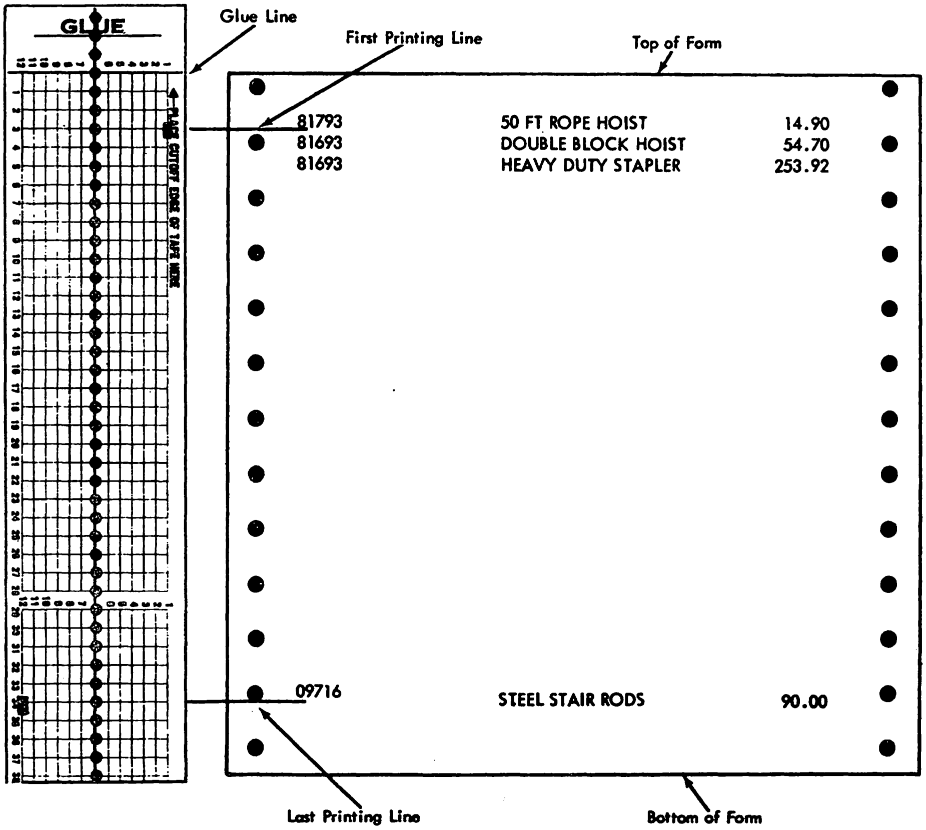

GLUE
PLACE CUTOFF EDGE OF TAPE HERE
Glue Line
First Printing Line
Top of Form
81793 50 FT ROPE HOIST 14.90
81693 DOUBLE BLOCK HOIST 54.70
81693 HEAVY DUTY STAPLER 253.92
09716 STEEL STAIR RODS 90.00
Last Printing Line
Bottom of Form

Figure 8-1 Carriage Control Tape

On the carriage control tape the horizontal lines represent spacing at 6 lines to the inch. The 12 vertical lines are called Channels and are used to control skipping on the printer. The functions of two of the most frequently used punches on the control tape are:

1. CHANNEL 1 - A punch in Channel 1 is normally used to indicate the first printing line stop, that is, where the first line is to be printed on the report.
2. CHANNEL12- A punch in Channel 12 is used to indicate the overflow location, that is, where the last line on which printing is to occur.

In the illustration in Figure 8-1 the punch in Channel 1 is used to indicate the first printing line stop, that is, where the first line will be printed. The punch in Channel 12 is used for "overflow control", indicating where the last printing line on the paper is to occur.

The normal length of the carriage control tape is 132 horizontal lines. When preparing the control tape, the punches (punch in Channel 1 and Channel 12) are normally repeated as many times as possible within the 132 possible lines.

The tape is then cut and glued to form a concentric circle and inserted into a special unit in the printer where the punches in the Channels are read by special brushes to control skipping.

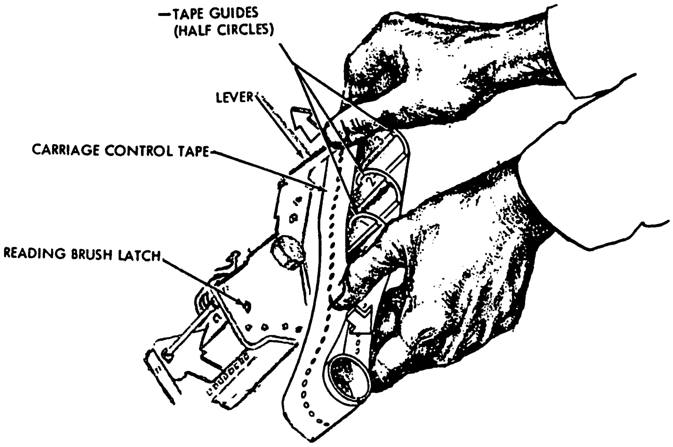
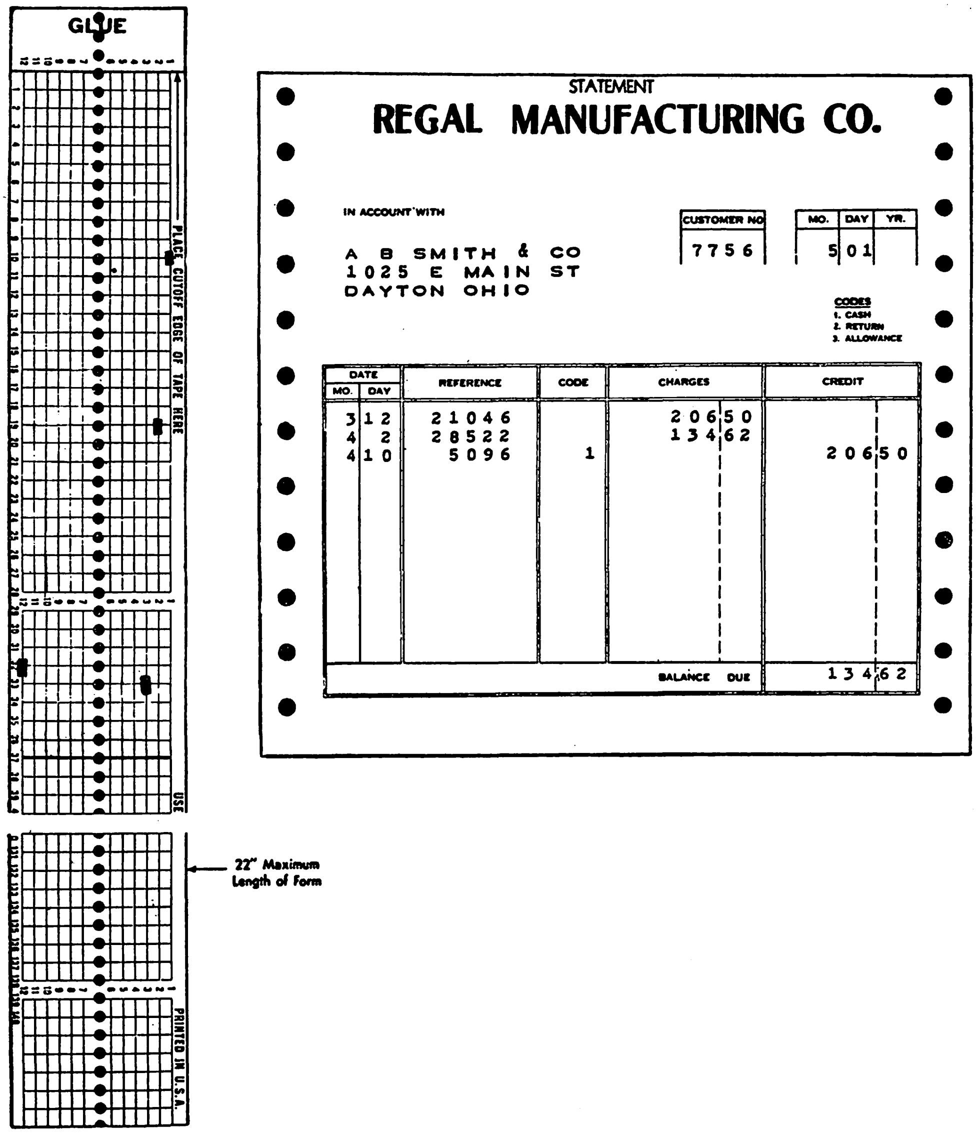


Figure 8-2 Inserting the Carriage Control Tape

Punches in the remaining Channels may also be used to control skipping. The following invoice and related control tape illustrate the use of the carriage control tape in the preparation of a typical invoice.



STATEMENT

REGAL MANUFACTURING CO.

IN ACCOUNT WITH

A B SMITH & CO
1025 E MAIN ST
DAYTON OHIO

CUSTOMER NO
7756

MO.	DAY	YR.
5	01	

CODES
1. CASH
2. RETURN
3. ALLOWANCE

DATE MO.	DATE DAY	REFERENCE	CODE	CHARGES	CREDIT
3	12	21046		206 50	
4	2	28522		134 62	
4	10	5096	1		206 50
				BALANCE DUE	134 62

Figure 8-3 Carriage Control Tape for Invoice Application

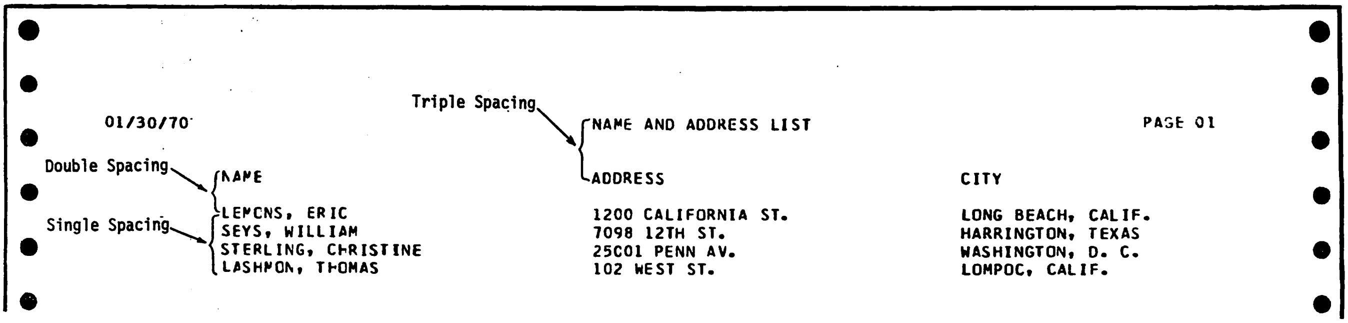
The punch in Channel 2 is used to identify where printing is to begin for the "body line", and the punch in Channel 3 to identify where printing is to occur for the printing of the total.

Note that the overflow punch on line 32 will prevent the printing of the body of the invoice on the Balance Due line.

It should be noted that the instructions to begin a skip are initiated by means of a Statement in the COBOL Program, and the termination of this skip is controlled by means of the carriage control tape.

HEADING CONTROL LOGIC

The sample problem developed in this chapter illustrates an operation in which a report title and column headings are to be printed on each page of a report. The computer is to print the report title at the top of each new page, triple space and print the column headings, double space and print the first detail line. The main body of the report is to be single spaced. There are to be fifty lines printed in the body of the report. After the 50th line is printed a "skip" is to occur so the next line (report title) prints on the first printing line of the next page as identified by the punch in Channel 1 of the carriage control tape. The following is an illustration of a segment of the required report. Note the spacing in the report heading.



```
   01/30/70                          NAME AND ADDRESS LIST                                   PAGE 01


          NAME                       ADDRESS                          CITY

          LEMONS, ERIC               1200 CALIFORNIA ST.              LONG BEACH, CALIF.
          SEYS, WILLIAM              7098 12TH ST.                    HARRINGTON, TEXAS
          STERLING, CHRISTINE        25C01 PENN AV.                   WASHINGTON, D. C.
          LASHMON, THOMAS            102 WEST ST.                     LOMPOC, CALIF.
```

Triple Spacing
Double Spacing
Single Spacing

Figure 8-4 Sample Report

The basic logic required in the solution of the problem is illustrated on the following page and explained below.

1. The 040-HEADING-ROUTINE is performed.
2. At the 040-HEADING-ROUTINE, the report title is printed after skipping to the first print line on a page. In addition, the page number is incremented by 1 prior to the printing of the heading line.
3. The column headings are printed after triple spacing.
4. A statement is given to set the space control in the WRITE statement used in the main routine so that double spacing will occur. This is accomplished by using an identifier in the AFTER ADVANCING option, and establishing a value for that identifier which will cause double spacing.
5. The line counter used to determine when all of the lines have been printed on a page is reset to zero.
6. The heading routine returns control to the main routine where a card is read and the print line is prepared.
7. A WRITE statement is executed. Double spacing will occur before the first detail line is printed (the Name, Address, City) as the spacing was set in step 4 of the 040-HEADING-ROUTINE to cause double spacing for the first line after the column headings.
8. When the first detail line after the column headings is printed, all subsequent lines are to be single spaced. Therefore, a statement is given to move a value which will cause single spacing to the identifier used in the AFTER ADVANCING to control spacing.

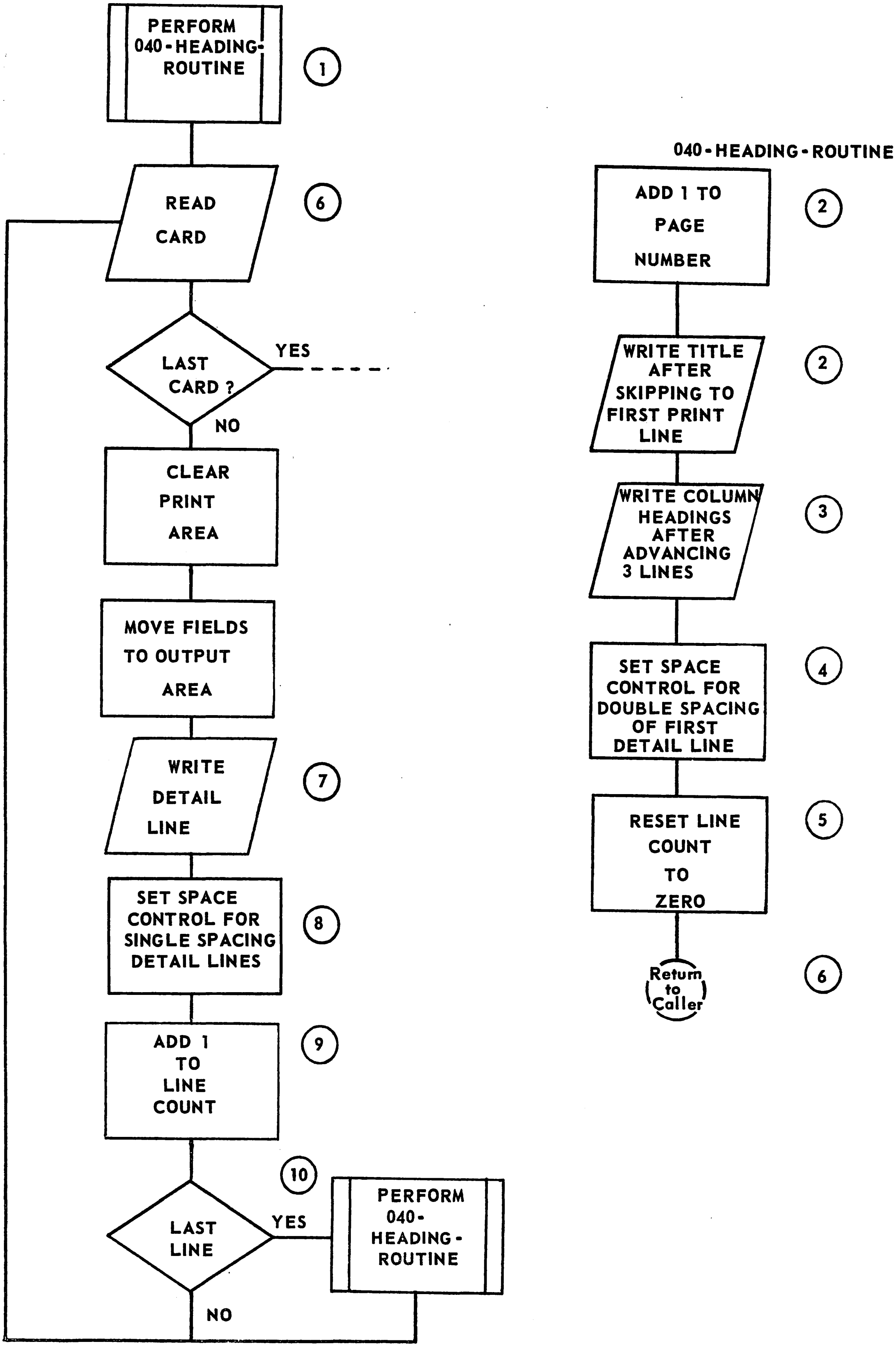

PERFORM 040-HEADING-ROUTINE
1
READ CARD
6
LAST CARD ?
YES
NO
CLEAR PRINT AREA
MOVE FIELDS TO OUTPUT AREA
WRITE DETAIL LINE
7
SET SPACE CONTROL FOR SINGLE SPACING DETAIL LINES
8
ADD 1 TO LINE COUNT
9
10
LAST LINE
YES
NO
PERFORM 040-HEADING-ROUTINE
040-HEADING-ROUTINE
ADD 1 TO PAGE NUMBER
2
WRITE TITLE AFTER SKIPPING TO FIRST PRINT LINE
2
WRITE COLUMN HEADINGS AFTER ADVANCING 3 LINES
3
SET SPACE CONTROL FOR DOUBLE SPACING OF FIRST DETAIL LINE
4
RESET LINE COUNT TO ZERO
5
Return to Caller
6

Figure 8-5 Program Logic for Heading Control

9. The value "1" is then added to the line counter indicating that one line has been printed on the new page. This counter will be incremented by one each time a detail line is printed until the value in the counter is equal to "50".

10. When the value in the counter is equal to "50", it indicates that the last line on the page has been printed, so the heading routine will again be executed to print a heading on the next page of the report.

SPECIAL-NAMES – Carriage Control

As noted previously, punches in the carriage control tape may be used to position the printer to the proper print line. In many instances, the character punched in channel 1 is used to indicate the first line to be printed on a page. Thus, the value following the AFTER ADVANCING clause in the WRITE statement must be identified to the compiler as the value meaning skip to channel one in the carriage control tape. This is done by placing the proper entry in the Special-Names paragraph in the Environment Division as illustrated below.

EXAMPLE

```
SPECIAL-NAMES.
    C01 IS TO-TOP-OF-PAGE.
```

Figure 8-6 Example of Special-Names for Carriage Control

Note in the example above that following the paragraph name SPECIAL-NAMES is the sentence "C01 IS TO-TOP-OF-PAGE". The "C01" is a special word which indicates that a symbolic name is to be given Channel 01 in the carriage control tape. The word "IS" is required and the identifier following is a programmer chosen identifier which is a symbolic name given to channel 1 in the carriage control tape. In this example, the chosen symbolic name is TO-TOP-OF-PAGE.

Whenever there is to be skipping to a punch in the carriage control tape, the desired punch must be defined with a symbolic name. Thus, if channel 11 in the carriage control tape were to be used, the entry "C11 IS . . ." would be included in the Special-Names paragraph to give a symbolic name to channel 11.

WRITE STATEMENT

Previous sample programs have illustrated the use of an integer with the AFTER ADVANCING option of the Write statements. It should be recalled that when the AFTER ADVANCING option is used, the integer must be unsigned and have a value less than 100. It will also be recalled that the Special-Names paragraph is used to apply a symbolic name to the punches in a carriage control tape. Thus, in the sample program, when a title line is to be printed as the first line on a new page and then the column headings are to be printed after triple spacing, the following Write statements may be utilized.

```
WRITE LINE-OUT FROM TITLE AFTER ADVANCING
    TO-TOP-OF-PAGE.
WRITE LINE-OUT FROM HEADING-LINE AFTER ADVANCING 3 LINES.
```

Figure 8-7 Example of WRITE Statements

Note from the example in Figure 8-7 that the symbolic name assigned in the example in Figure 8-6 is used following the AFTER Advancing to indicate that the printer is to be skipped to channel 1 in the carriage control tape prior to printing the TITLE line. After the first line is written, the printer is advanced three lines before printing the column headings.

A data-identifier may also be specified in the AFTER ADVANCING option of the WRITE statement. The following is an illustration of the format notation of the WRITE statement.

```
WRITE record-name [FROM identifier-1]

[ { BEFORE }               { identifier-2 LINES } ]
[ {        }   ADVANCING   { integer LINES      } ]
[ { AFTER  }               { mnemonic-name      } ]
```

Figure 8-8 Format Notation – WRITE Statement

Note from Figure 8-8 that identifier-2 may be used to specify the name of a nonnegative numeric elementary item described as an integer which contains a value of less than 100. When identifier-2 is specified, the printer page is advanced the number of lines contained in the identifier. Thus, if the value 2 is contained in identifier-2, the printer will be advanced 2 lines.

The following is an example of a WRITE statement using an identifier with the AFTER ADVANCING option.

```
WRITE LINE-OUT FROM ADDRESS-LIST
    AFTER ADVANCING VARIABLE LINES.
```

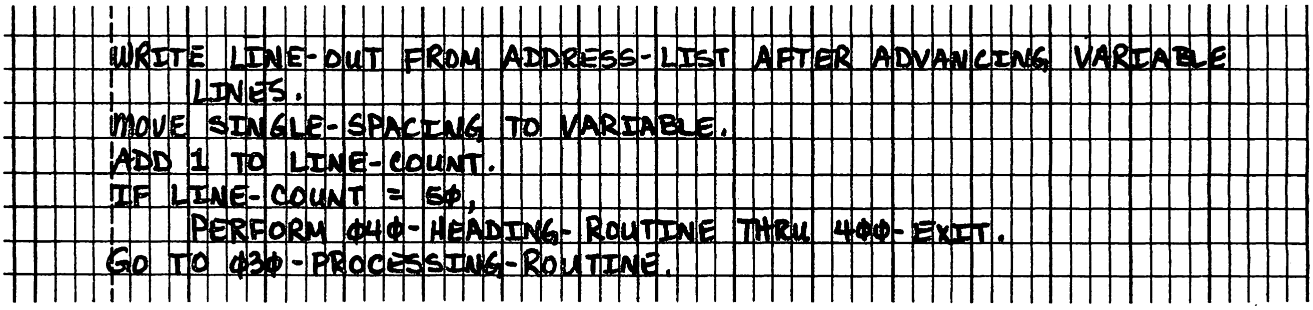
Figure 8-9 Example of Write Statement with Identifier

If the value referenced by the Identifier VARIABLE is "1", the printer will be spaced one line. If VARIABLE assumes a value of "2", double spacing will occur. The values which VARIABLE will assume in this example are set up in WORKING-STORAGE as a Level 77 item with a VALUE clause.

Thus, if a "2" is moved to VARIABLE, when the WRITE statement is executed, double spacing will occur. If a "1" is moved to VARIABLE, single spacing will take place when the WRITE statement is executed.

LINE COUNTING

When a page is being printed, it is necessary to be able to determine when all of the lines on the page have been printed and it is time to skip to the next page and print headings on the new page before continuing with the printing of the detail data. This determination can be made by counting the number of lines which have been printed on a given page and when the desired number of lines have been printed, the heading routine can be performed prior to printing any more detail lines. The method of line counting used in the sample program is illustrated below.



```
WRITE LINE-OUT FROM ADDRESS-LIST AFTER ADVANCING VARIABLE
    LINES.
MOVE SINGLE-SPACING TO VARIABLE.
ADD 1 TO LINE-COUNT.
IF LINE-COUNT = 50,
    PERFORM 040-HEADING-ROUTINE THRU 400-EXIT.
GO TO 030-PROCESSING-ROUTINE.
```

Figure 8-10 Example of Line Counting

In the example above it can be seen that a line is printed on the report. The value "1", which is contained in the field SINGLE-SPACING, is then moved to the field VARIABLE so that single spacing will occur. The value "1" is then added to the field LINE-COUNT, which keeps track of the number of lines printed on the page. If the value in LINE-COUNT has reached 50, it indicates that 50 lines have been printed on the page. Since this is the desired number of lines to be printed on a single page, the 040-HEADING-ROUTINE will be performed. If LINE-COUNT is not equal to 50, then control will pass back to read the next detail card and process another line of detail data.

It should be noted that each time a heading is to be printed on the report, the value in LINE-COUNT should be reset to zero so that a proper line count will be kept for the new page. This is accomplished in the 040-HEADING-ROUTINE.

PERFORM STATEMENT

To control the printing of headings on a report a SUBROUTINE is normally used. A subroutine may be defined as a "standardized set of instructions which direct the computer to execute a particular operation". Subroutines are used where the same set of instructions are required at several different points in the program.

Basically there are two types of subroutines, an OPEN Subroutine and a CLOSED Subroutine.

An Open Subroutine is one that is inserted directly in the main line of the program whenever needed. If the same sequence of instructions are required several times in the program, the Open Subroutine must be recopied at each point in the program.

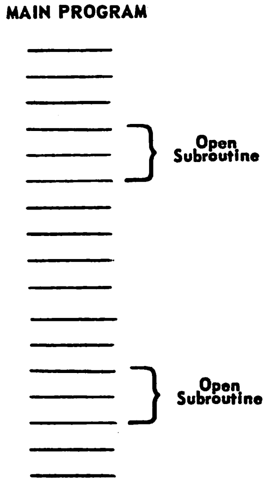


Figure 8-11 Open Subroutine

The Closed Subroutine is a set of instructions required by the program that is not stored in the main line of the program, and may be utilized repeatedly when needed in the main program.

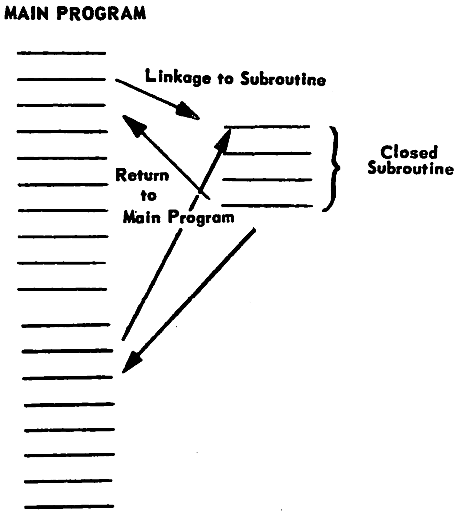


Figure 8-12 Closed Subroutine

This type of subroutine requires some method or programming technique to provide a LINKAGE to the subroutine, and some method of return to the next instruction in sequence in the main program. In COBOL the PERFORM Statement is normally used to provide linkage to the subroutine and automatic return to the next instruction in the main program.

The PERFORM Statement specifies a transfer of control from one portion of a program to another, in order to execute some procedure a specified number of times or until a condition is satisfied. It directs that control is to be returned to the statement immediately following the point from which the transfer was made.

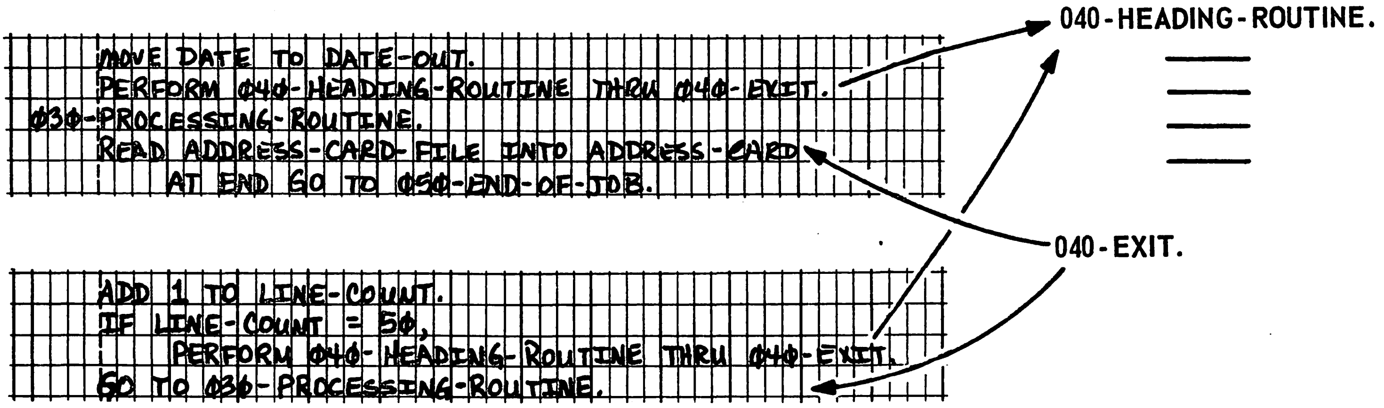
The PERFORM Statement has the following basic format:

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

Figure 8-13 Format Notation – PERFORM Statement

A procedure referred to by this type of PERFORM Statement is executed once, and then control passes to the next statement AFTER the PERFORM Statement. All statements in the paragraphs or sections named by procedure-name-1 through procedure-name-2 constitute the range of the PERFORM Statement.

In the sample flowchart illustrated on page 8.5 note that 040-HEADING-ROUTINE is performed at the beginning of the program, and after 50 lines have been printed on a single page. To execute the 040-HEADING-ROUTINE at both points in the program and to return to the next sequential instruction, the PERFORM statement is used. The following segment of the COBOL program illustrates the use of the PERFORM.



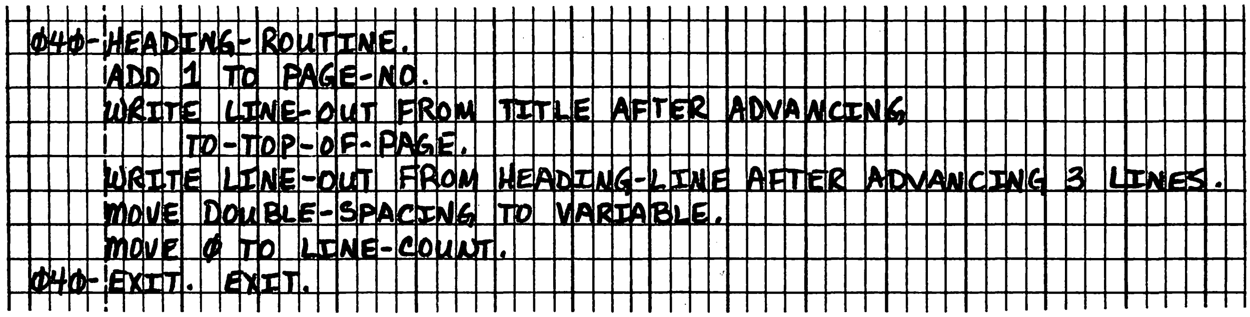
```
    MOVE DATE TO DATE-OUT.
    PERFORM 040-HEADING-ROUTINE THRU 040-EXIT.
030-PROCESSING-ROUTINE.
    READ ADDRESS-CARD-FILE INTO ADDRESS-CARD
        AT END GO TO 050-END-OF-JOB.
```

```
    ADD 1 TO LINE-COUNT.
    IF LINE-COUNT = 50,
        PERFORM 040-HEADING-ROUTINE THRU 040-EXIT.
    GO TO 030-PROCESSING-ROUTINE.
```

Figure 8-14 Segment of Procedure Division illustrating use of the PERFORM Statement

In the example above note the use of the PERFORM statement at two different points in the program. When the statement PERFORM 040-HEADING-ROUTINE is executed, the instructions at 040-HEADING-ROUTINE will be executed then the program will return to the main routine and execute the instruction following the PERFORM.

It should be noted that both PERFORM statements contain the entry "THRU 040-EXIT". This indicates that all paragraphs beginning with the paragraph-name 040-HEADING-ROUTINE through the paragraph-name 040-EXIT will be executed as a closed subroutine. The heading subroutine used in the sample program is illustrated in Figure 8-15.



```
040-HEADING-ROUTINE.
    ADD 1 TO PAGE-NO.
    WRITE LINE-OUT FROM TITLE AFTER ADVANCING
           TO-TOP-OF-PAGE.
    WRITE LINE-OUT FROM HEADING-LINE AFTER ADVANCING 3 LINES.
    MOVE DOUBLE-SPACING TO VARIABLE.
    MOVE 0 TO LINE-COUNT.
040-EXIT. EXIT.
```

Figure 8-15 Example of HEADING ROUTINE

Note in the example above that the paragraph begins with the paragraph-name 040-HEADING-ROUTINE. Thus, when the PERFORM statement is executed, the first statement to be executed in the subroutine will be the ADD statement which increments the page number by 1 prior to printing it in the heading.

The remainder of the statements in the paragraph will be executed in a sequential manner. When control comes to the EXIT statement which is contained in the paragraph 040-EXIT, control will be returned to calling routine, that is, the statement following the PERFORM statement which transferred control to the subroutine.

The EXIT statement may be used at the conclusion of a performed subroutine to transfer control back to the main routine. It is not required because control will always be passed back when the indicated paragraphs in the PERFORM statement have been executed. It is normally good programming technique, however, to have the EXIT statement as the last statement in a performed subroutine to indicate the exact point at which processing within the performed subroutine will be completed. The EXIT statement must be written in conjunction with a paragraph-name and must be the only statement within the paragraph. The format notation of the EXIT statement is illustrated in Figure 8-16.

```
paragraph-name.  EXIT.
```

Figure 8-16 Format Notation – EXIT Statement

Note from Figure 8-16 that a paragraph name, such as 040-EXIT, must be used and the only statement within the paragraph is the word EXIT. It should be noted also that when the EXIT statement is used as the last statement within a subroutine, the paragraph-name used in conjunction with the EXIT statement should be the paragraph-name specified in the "THRU" optional portion of the PERFORM statement, as illustrated in Figure 8-14.

PRINTING THE CURRENT DATE IN REPORT HEADINGS

The sample problem developed in this chapter illustrates one method for the printing and controlling of report and column headings. In addition to the printing of report and column headings, the current date and the page number are to be printed on the same line as the report title.

One technique for the printing of the current date on a report is to utilize a "Date Card". This card is keypunched with the current date prior to running the report and is placed as the first card of the data to be processed.

In the sample problem, the Date Card contains a "D" control punch in card column one. The control punch is followed by two digits for the month, a slash, two digits for the day, a slash, and two digits for the year. For example, the date January 30, 1970 would be punched as 01/30/70 in card columns 2 through 9. The following segment of the sample program illustrates the routine to read the date card.

```
020-READ-DATE.
    READ ADDRESS-CARD-FILE INTO DATE-CARD
        AT END GO TO 050-END-OF-JOB.
    IF CODE-FIELD NOT = 'D' STOP 'INCORRECT DATE CARD'
        GO TO 020-READ-DATE.
    MOVE DATE TO DATE-OUT.
    PERFORM 040-HEADING-ROUTINE THRU 040-EXIT.
```

Figure 8-17 "Date" Routine

From the statements above it can be seen that if the first card in the data deck does not contain a "D" punch, a message is printed on the console and processing pauses. This gives the operator the opportunity to prepare the date card and to begin the operation again. If the card contains a "D" control punch, the "DATE" is moved to the output line and the heading routine is executed.

SAMPLE PROBLEM

The following diagrams illustrate the complete documentation for a problem which includes report and column headings, testing for page overflow, printing of the current date on the heading of the report, and the numbering of each page of the report.

Note in the sample COBOL program in Figure 8-21 that an input area for the Date Card and an input area for the Name and Address cards is defined in Working Storage. In the Data Division only the general format of the two input records is defined (see Line 29, Figure 8-21). The "Read Into" option is required when this technique for defining input records is used. For example, to read the Date Card, the statement READ ADDRESS-CARD-FILE INTO DATE-CARD is specified (see Line 81 of the sample COBOL program in Figure 8-21.

The output area for the detail printed lines is also specified in Working Storage in this example. These detail lines are written by use of the Write statement with the "From" option. (See Lines 94, 103, and 105 in the sample COBOL program in Figure 8-21.

Input

Input consists of a Date Card followed by Name and Address Cards which contain a Name Field, an Address Field, and a City Field. Output consists of a printed report. The input, output, flowchart, and program are illustrated on the following pages.

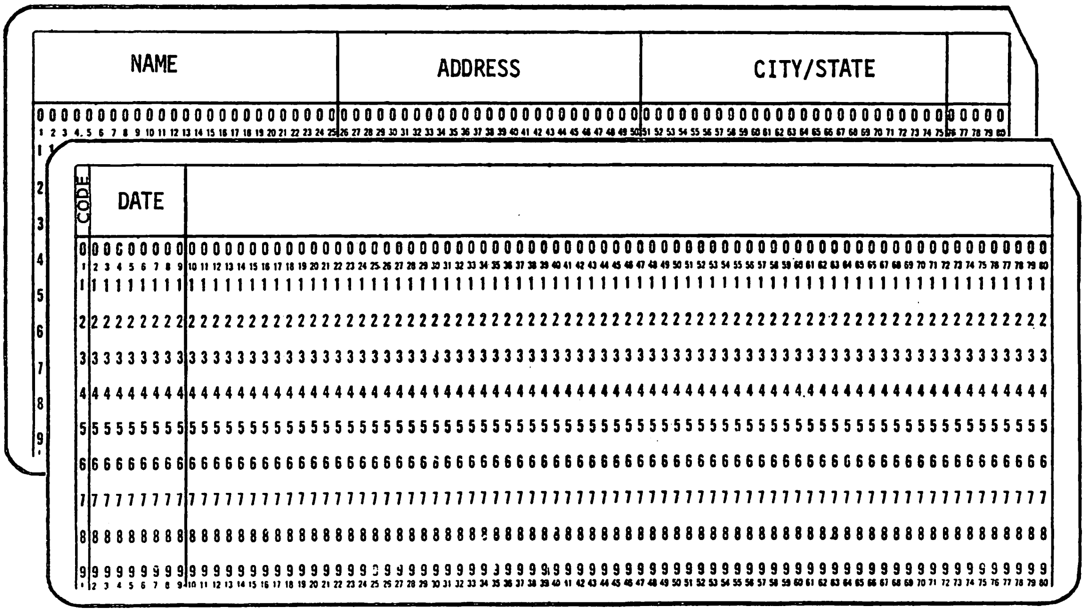
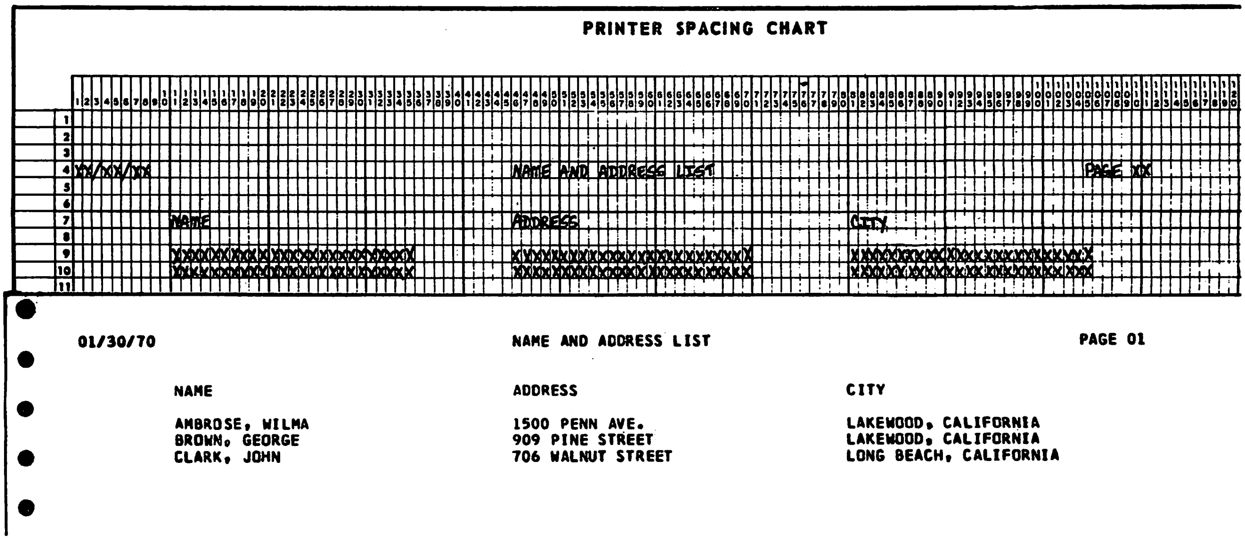


Figure 8-18 Input

Output



PRINTER SPACING CHART

```
XX/XX/XX                                     NAME AND ADDRESS LIST                                     PAGE XX


          NAME                               ADDRESS                          CITY

          XXXXXXXXXXXXXXXXXXXXXXXXX          XXXXXXXXXXXXXXXXXXXXXXXXX        XXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXX          XXXXXXXXXXXXXXXXXXXXXXXXX        XXXXXXXXXXXXXXXXXXXXXXXXX
```

```
01/30/70                                     NAME AND ADDRESS LIST                                     PAGE 01

          NAME                               ADDRESS                          CITY

          AMBROSE, WILMA                     1500 PENN AVE.                   LAKEWOOD, CALIFORNIA
          BROWN, GEORGE                      909 PINE STREET                  LAKEWOOD, CALIFORNIA
          CLARK, JOHN                        706 WALNUT STREET                LONG BEACH, CALIFORNIA
```

Figure 8-19 Output

Flowchart

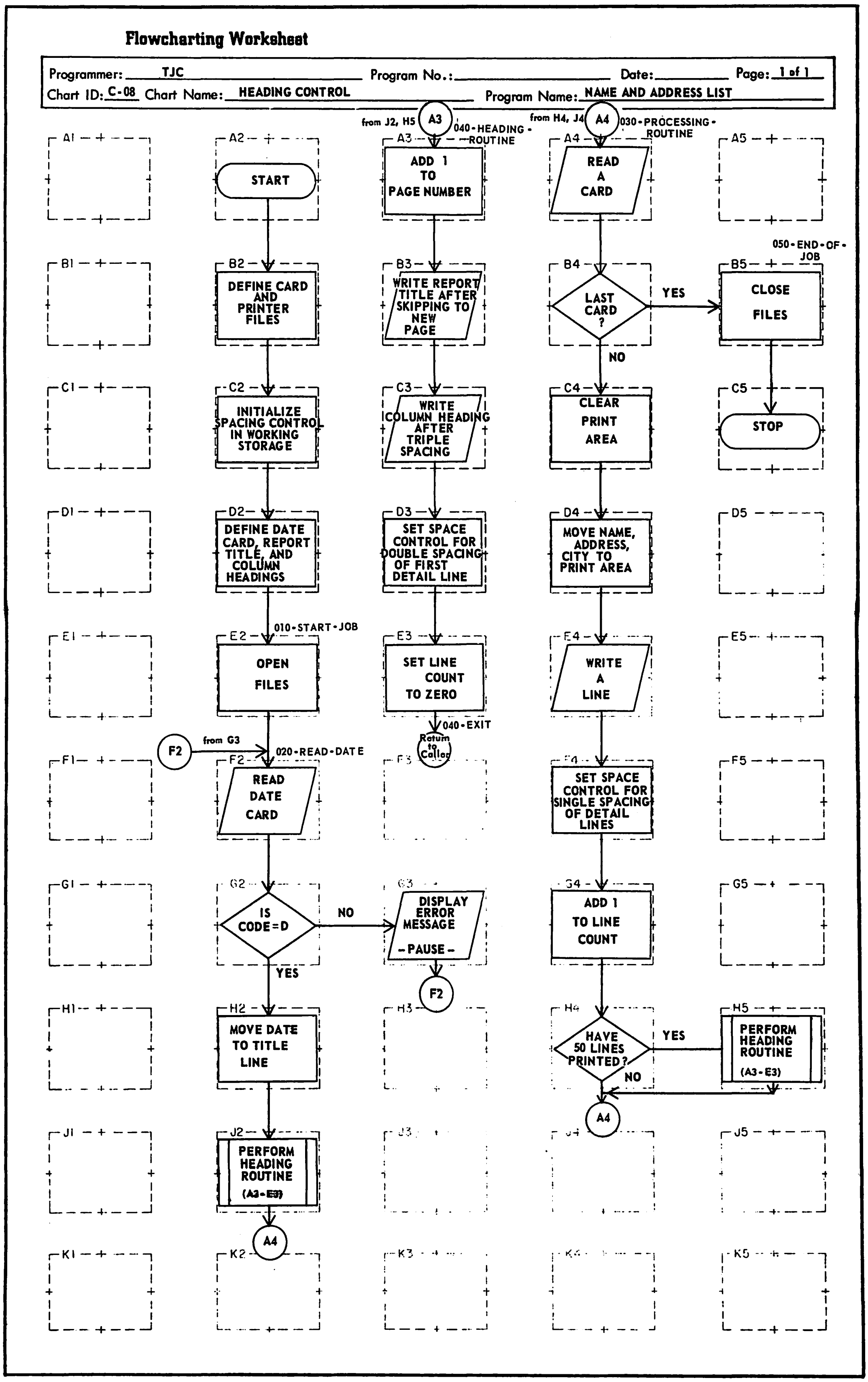
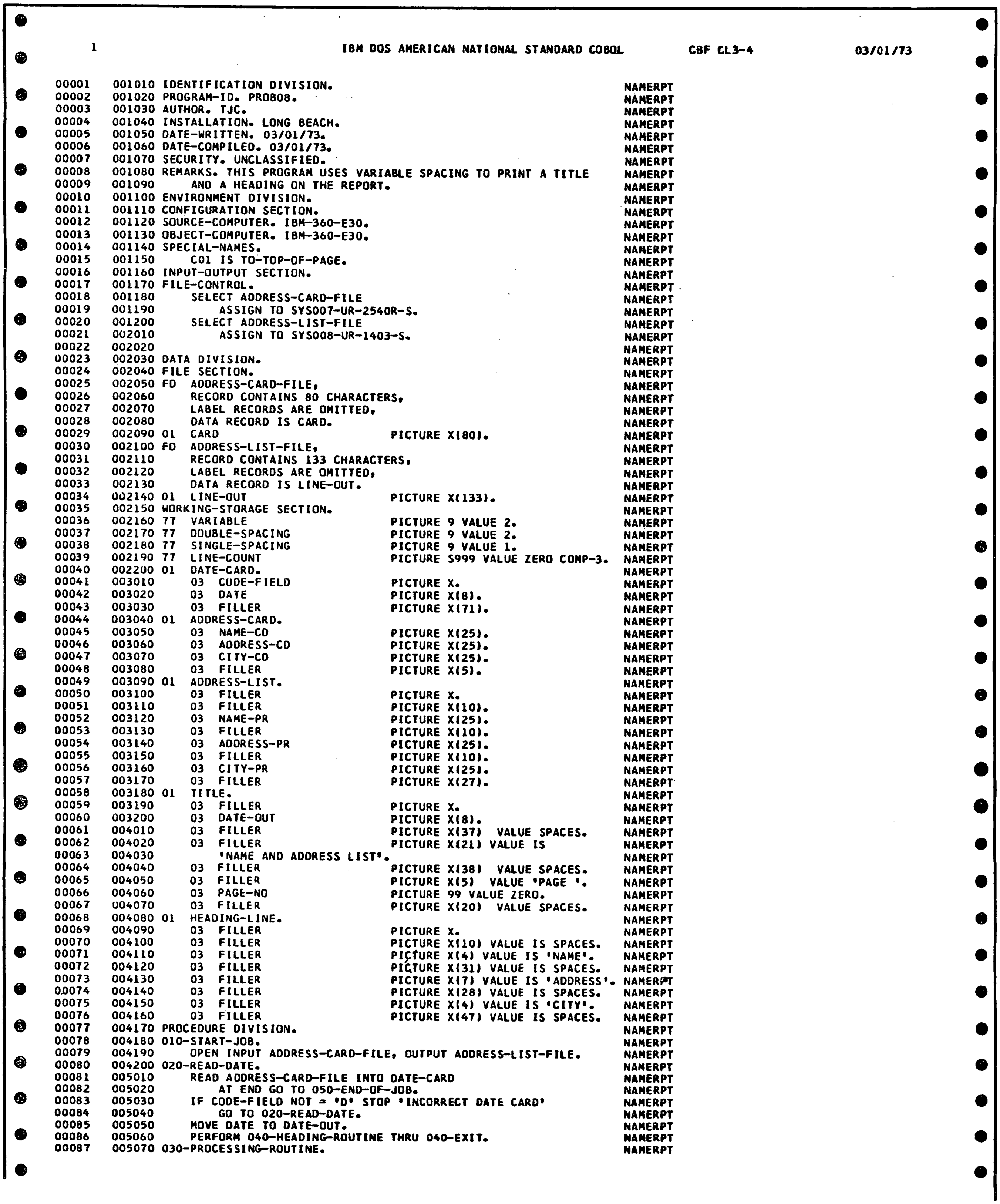

Flowcharting Worksheet
Programmer: TJC
Program No.:
Date:
Page: 1 of 1
Chart ID: C-08
Chart Name: HEADING CONTROL
Program Name: NAME AND ADDRESS LIST
START
DEFINE CARD AND PRINTER FILES
INITIALIZE SPACING CONTROL IN WORKING STORAGE
DEFINE DATE CARD, REPORT TITLE, AND COLUMN HEADINGS
010-START-JOB
OPEN FILES
F2
from G3
020-READ-DATE
READ DATE CARD
IS CODE=D
NO
YES
MOVE DATE TO TITLE LINE
PERFORM HEADING ROUTINE (A3-E3)
A4
from J2, H5
A3
040-HEADING-ROUTINE
ADD 1 TO PAGE NUMBER
WRITE REPORT TITLE AFTER SKIPPING TO NEW PAGE
WRITE COLUMN HEADING AFTER TRIPLE SPACING
SET SPACE CONTROL FOR DOUBLE SPACING OF FIRST DETAIL LINE
SET LINE COUNT TO ZERO
040-EXIT
Return to Caller
DISPLAY ERROR MESSAGE -PAUSE-
F2
from H4, J4
A4
030-PROCESSING-ROUTINE
READ A CARD
LAST CARD ?
YES
NO
CLEAR PRINT AREA
MOVE NAME, ADDRESS, CITY TO PRINT AREA
WRITE A LINE
SET SPACE CONTROL FOR SINGLE SPACING OF DETAIL LINES
ADD 1 TO LINE COUNT
HAVE 50 LINES PRINTED?
YES
NO
A4
PERFORM HEADING ROUTINE (A3-E3)
050-END-OF-JOB
CLOSE FILES
STOP

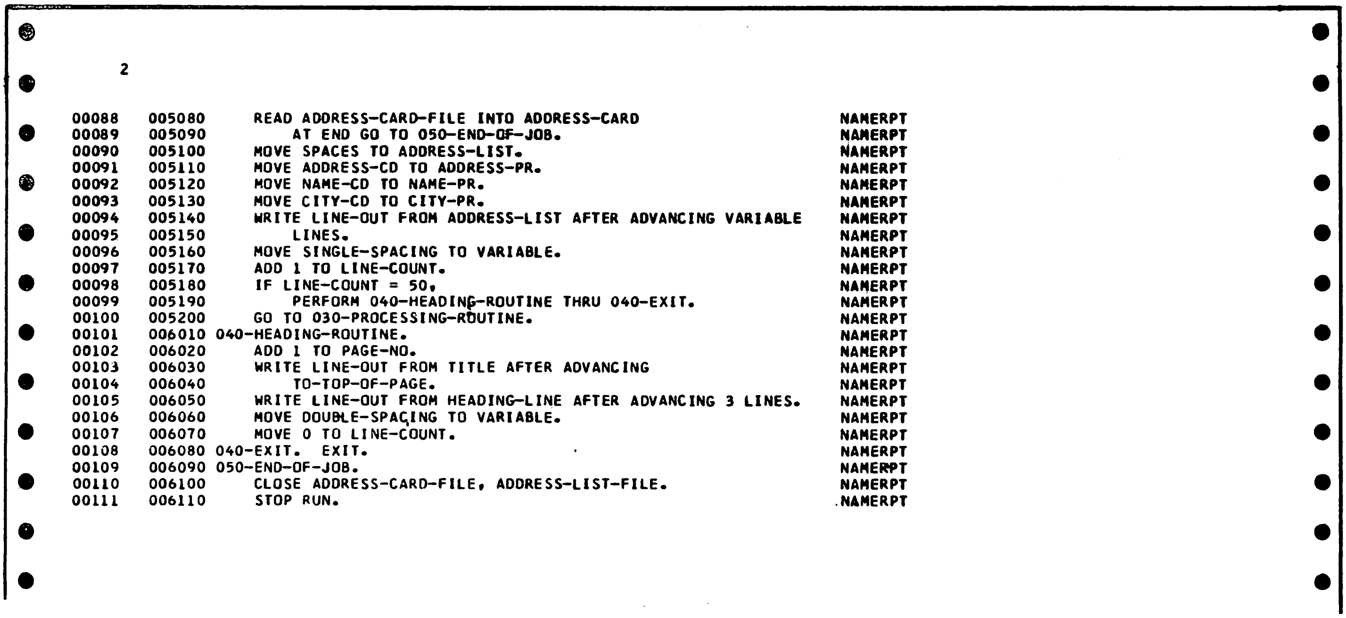
Figure 8-20 Flowchart

COBOL Program



```
          1                                          IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4           03/01/73

00001   001010 IDENTIFICATION DIVISION.                                         NAMERPT
00002   001020 PROGRAM-ID. PRO808.                                              NAMERPT
00003   001030 AUTHOR. TJC.                                                     NAMERPT
00004   001040 INSTALLATION. LONG BEACH.                                        NAMERPT
00005   001050 DATE-WRITTEN. 03/01/73.                                          NAMERPT
00006   001060 DATE-COMPILED. 03/01/73.                                         NAMERPT
00007   001070 SECURITY. UNCLASSIFIED.                                          NAMERPT
00008   001080 REMARKS. THIS PROGRAM USES VARIABLE SPACING TO PRINT A TITLE     NAMERPT
00009   001090     AND A HEADING ON THE REPORT.                                 NAMERPT
00010   001100 ENVIRONMENT DIVISION.                                            NAMERPT
00011   001110 CONFIGURATION SECTION.                                           NAMERPT
00012   001120 SOURCE-COMPUTER. IBM-360-E30.                                    NAMERPT
00013   001130 OBJECT-COMPUTER. IBM-360-E30.                                    NAMERPT
00014   001140 SPECIAL-NAMES.                                                   NAMERPT
00015   001150     C01 IS TO-TOP-OF-PAGE.                                       NAMERPT
00016   001160 INPUT-OUTPUT SECTION.                                            NAMERPT
00017   001170 FILE-CONTROL.                                                    NAMERPT
00018   001180     SELECT ADDRESS-CARD-FILE                                     NAMERPT
00019   001190         ASSIGN TO SYS007-UR-2540R-S.                             NAMERPT
00020   001200     SELECT ADDRESS-LIST-FILE                                     NAMERPT
00021   002010         ASSIGN TO SYS008-UR-1403-S.                              NAMERPT
00022   002020                                                                  NAMERPT
00023   002030 DATA DIVISION.                                                   NAMERPT
00024   002040 FILE SECTION.                                                    NAMERPT
00025   002050 FD  ADDRESS-CARD-FILE,                                           NAMERPT
00026   002060     RECORD CONTAINS 80 CHARACTERS,                               NAMERPT
00027   002070     LABEL RECORDS ARE OMITTED,                                   NAMERPT
00028   002080     DATA RECORD IS CARD.                                         NAMERPT
00029   002090 01  CARD                            PICTURE X(80).               NAMERPT
00030   002100 FD  ADDRESS-LIST-FILE,                                           NAMERPT
00031   002110     RECORD CONTAINS 133 CHARACTERS,                              NAMERPT
00032   002120     LABEL RECORDS ARE OMITTED,                                   NAMERPT
00033   002130     DATA RECORD IS LINE-OUT.                                     NAMERPT
00034   002140 01  LINE-OUT                        PICTURE X(133).              NAMERPT
00035   002150 WORKING-STORAGE SECTION.                                         NAMERPT
00036   002160 77  VARIABLE                        PICTURE 9 VALUE 2.           NAMERPT
00037   002170 77  DOUBLE-SPACING                  PICTURE 9 VALUE 2.           NAMERPT
00038   002180 77  SINGLE-SPACING                  PICTURE 9 VALUE 1.           NAMERPT
00039   002190 77  LINE-COUNT                      PICTURE S999 VALUE ZERO COMP-3.  NAMERPT
00040   002200 01  DATE-CARD.                                                   NAMERPT
00041   003010     03  CODE-FIELD                  PICTURE X.                   NAMERPT
00042   003020     03  DATE                        PICTURE X(8).                NAMERPT
00043   003030     03  FILLER                      PICTURE X(71).               NAMERPT
00044   003040 01  ADDRESS-CARD.                                                NAMERPT
00045   003050     03  NAME-CD                     PICTURE X(25).               NAMERPT
00046   003060     03  ADDRESS-CD                  PICTURE X(25).               NAMERPT
00047   003070     03  CITY-CD                     PICTURE X(25).               NAMERPT
00048   003080     03  FILLER                      PICTURE X(5).                NAMERPT
00049   003090 01  ADDRESS-LIST.                                                NAMERPT
00050   003100     03  FILLER                      PICTURE X.                   NAMERPT
00051   003110     03  FILLER                      PICTURE X(10).               NAMERPT
00052   003120     03  NAME-PR                     PICTURE X(25).               NAMERPT
00053   003130     03  FILLER                      PICTURE X(10).               NAMERPT
00054   003140     03  ADDRESS-PR                  PICTURE X(25).               NAMERPT
00055   003150     03  FILLER                      PICTURE X(10).               NAMERPT
00056   003160     03  CITY-PR                     PICTURE X(25).               NAMERPT
00057   003170     03  FILLER                      PICTURE X(27).               NAMERPT
00058   003180 01  TITLE.                                                       NAMERPT
00059   003190     03  FILLER                      PICTURE X.                   NAMERPT
00060   003200     03  DATE-OUT                    PICTURE X(8).                NAMERPT
00061   004010     03  FILLER                      PICTURE X(37)  VALUE SPACES.     NAMERPT
00062   004020     03  FILLER                      PICTURE X(21) VALUE IS           NAMERPT
00063   004030         'NAME AND ADDRESS LIST'.                                 NAMERPT
00064   004040     03  FILLER                      PICTURE X(38)  VALUE SPACES.     NAMERPT
00065   004050     03  FILLER                      PICTURE X(5)  VALUE 'PAGE '.     NAMERPT
00066   004060     03  PAGE-NO                     PICTURE 99 VALUE ZERO.           NAMERPT
00067   004070     03  FILLER                      PICTURE X(20)  VALUE SPACES.     NAMERPT
00068   004080 01  HEADING-LINE.                                                NAMERPT
00069   004090     03  FILLER                      PICTURE X.                   NAMERPT
00070   004100     03  FILLER                      PICTURE X(10) VALUE IS SPACES.   NAMERPT
00071   004110     03  FILLER                      PICTURE X(4) VALUE IS 'NAME'.    NAMERPT
00072   004120     03  FILLER                      PICTURE X(31) VALUE IS SPACES.   NAMERPT
00073   004130     03  FILLER                      PICTURE X(7) VALUE IS 'ADDRESS'. NAMERPT
00074   004140     03  FILLER                      PICTURE X(28) VALUE IS SPACES.   NAMERPT
00075   004150     03  FILLER                      PICTURE X(4) VALUE IS 'CITY'.    NAMERPT
00076   004160     03  FILLER                      PICTURE X(47) VALUE IS SPACES.   NAMERPT
00077   004170 PROCEDURE DIVISION.                                              NAMERPT
00078   004180 010-START-JOB.                                                   NAMERPT
00079   004190     OPEN INPUT ADDRESS-CARD-FILE, OUTPUT ADDRESS-LIST-FILE.      NAMERPT
00080   004200 020-READ-DATE.                                                   NAMERPT
00081   005010     READ ADDRESS-CARD-FILE INTO DATE-CARD                        NAMERPT
00082   005020         AT END GO TO 050-END-OF-JOB.                             NAMERPT
00083   005030     IF CODE-FIELD NOT = 'D' STOP 'INCORRECT DATE CARD'           NAMERPT
00084   005040         GO TO 020-READ-DATE.                                     NAMERPT
00085   005050     MOVE DATE TO DATE-OUT.                                       NAMERPT
00086   005060     PERFORM 040-HEADING-ROUTINE THRU 040-EXIT.                   NAMERPT
00087   005070 030-PROCESSING-ROUTINE.                                          NAMERPT
```

Figure 8-21 COBOL Program – Part 1 of 2



```
       2

00088   005080      READ ADDRESS-CARD-FILE INTO ADDRESS-CARD                    NAMERPT
00089   005090          AT END GO TO 050-END-OF-JOB.                            NAMERPT
00090   005100      MOVE SPACES TO ADDRESS-LIST.                                NAMERPT
00091   005110      MOVE ADDRESS-CD TO ADDRESS-PR.                              NAMERPT
00092   005120      MOVE NAME-CD TO NAME-PR.                                    NAMERPT
00093   005130      MOVE CITY-CD TO CITY-PR.                                    NAMERPT
00094   005140      WRITE LINE-OUT FROM ADDRESS-LIST AFTER ADVANCING VARIABLE   NAMERPT
00095   005150          LINES.                                                  NAMERPT
00096   005160      MOVE SINGLE-SPACING TO VARIABLE.                            NAMERPT
00097   005170      ADD 1 TO LINE-COUNT.                                        NAMERPT
00098   005180      IF LINE-COUNT = 50,                                         NAMERPT
00099   005190          PERFORM 040-HEADING-ROUTINE THRU 040-EXIT.              NAMERPT
00100   005200      GO TO 030-PROCESSING-ROUTINE.                               NAMERPT
00101   006010 040-HEADING-ROUTINE.                                             NAMERPT
00102   006020      ADD 1 TO PAGE-NO.                                           NAMERPT
00103   006030      WRITE LINE-OUT FROM TITLE AFTER ADVANCING                   NAMERPT
00104   006040          TO-TOP-OF-PAGE.                                         NAMERPT
00105   006050      WRITE LINE-OUT FROM HEADING-LINE AFTER ADVANCING 3 LINES.   NAMERPT
00106   006060      MOVE DOUBLE-SPACING TO VARIABLE.                            NAMERPT
00107   006070      MOVE 0 TO LINE-COUNT.                                       NAMERPT
00108   006080 040-EXIT.  EXIT.                                                 NAMERPT
00109   006090 050-END-OF-JOB.                                                  NAMERPT
00110   006100      CLOSE ADDRESS-CARD-FILE, ADDRESS-LIST-FILE.                 NAMERPT
00111   006110      STOP RUN.                                                   NAMERPT
```

Figure 8-22 COBOL Program – Part 2 of 2

CHAPTER 8

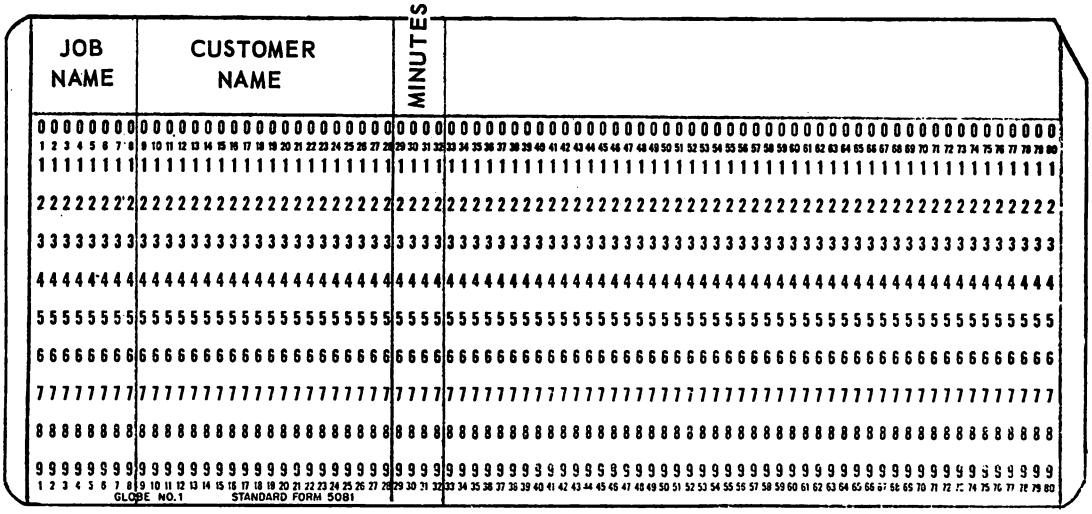
PROGRAMMING ASSIGNMENT 1

INSTRUCTIONS

A computer billing report is to be produced from card input which specifies the number of minutes which the computer is used for a given application. The billing rate for the computer is $50.00 per hour of usage.

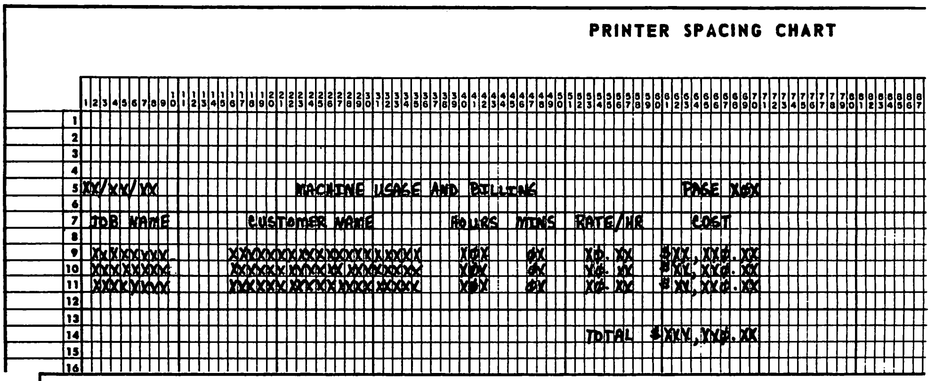
INPUT

The input consists of a card containing the Job Name, the Customer Name to which the job is to be billed, and the number of minutes which the job used on the computer.


JOB NAME
CUSTOMER NAME
MINUTES
GLOBE NO.1
STANDARD FORM 5081

OUTPUT

The output is to consist of the Computer Billing Report. The fields on the report are the Job Name, the Customer Name, the Hours and Minutes which the computer job used, the cost per hour, and the Total Billing Cost. Note that the minutes which are contained in the input card must be converted to hours and minutes in order to use the per hour billing rate and also to be printed on the report.


PRINTER SPACING CHART
XX/XX/XX
MACHINE USAGE AND BILLING
PAGE XXX
JOB NAME
CUSTOMER NAME
HOURS
MINS
RATE/HR
COST
XXXXXXXX
XXXXXXXXXXXXXXXXXXXX
XXX
XX
XX.XX
$XX,XXX.XX
TOTAL
$XXX,XXX.XX

```
03/22/73              MACHINE  USAGE  AND  BILLING                PAGE   1

 JOB  NAME          CUSTOMER  NAME          HOURS   MINS    RATE/HR        COST

 P792301A          ABSCO, INC.                  3     30      50.00     $    175.00
 P821024C          JACKSON'S CO.               20     36      50.00     $  1,030.00
 Q903478B          NELSON ELECTRONICS           7     42      50.00     $    385.00

                                                              TOTAL     $ 98,052.61
```

TEST DATA – ASSIGNMENT NO. 1

Job Name (col 1-8)	Customer Name (col 9-28)	Minutes (col 29-32)
P782301A	Absco, Inc.	0210
P821024C	Jackson's Co.	1236
Q903478D	Nelson Electronics	0462
S532671O	Quarry Gloves	0973
S853421B	Simmons Inc.	2896
T098876R	Timonson, Inc.	0032

CHAPTER 8

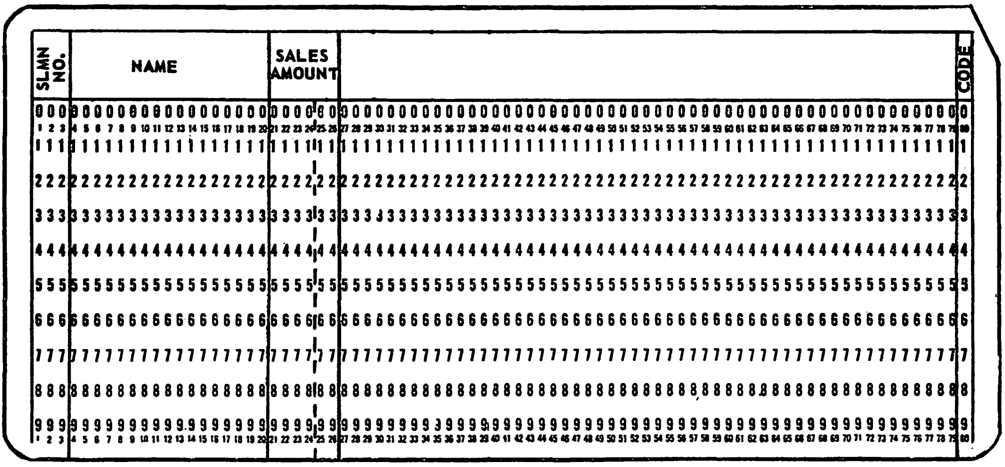
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

Write the COBOL program to produce a Sales Commission Report. Input cards will contain a control code to indicate the type of employee. A "1" code indicates a part-time employee. No commission is paid to part-time employees. A "2" punch indicates a fulltime salaried employee. Fulltime salaried employees receive a 5% commission on all sales. A "3" punch indicates employees working on a commission only. Employees working on a commission only basis receive a 5% commission on all sales up to $1000.00 and a 7% commission on all sales in excess of $1000.00. For example, a "commission only" employee with sales of $1200.00 would receive a commission of 5% on $1000.00 and a 7% commission on $200.00.

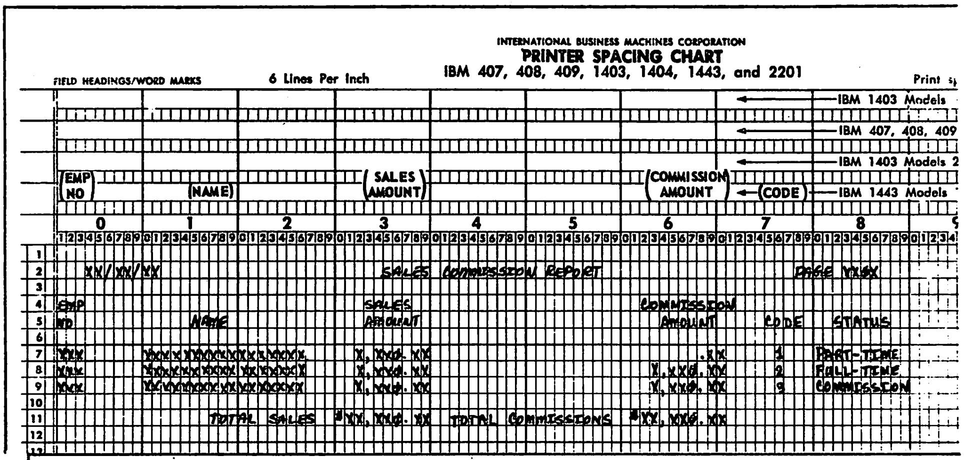
INPUT: Sales Cards

Input is to consist of Sales Cards containing the Salesman Number, Salesman Name, Sales Amount, and a Code. The format of the Sales Cards is illustrated below.



OUTPUT: Sales Commission Report

Output is to consist of a Sales Commission Report. The report is to contain the Salesman Number, Salesman Name, Sales Amount, and Commission. Employee cards which contain a "1" punch are to be identified as PART-TIME on the report. Employee cards containing a "2" punch are to be identified as FULL-TIME on the report and cards containing a "3" punch are to be identified as COMMISSION on the report. Final Totals are to be taken of the Sales Amount and Commission Amount.


INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, and 2201
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
Print
IBM 1403 Models
IBM 407, 408, 409
IBM 1403 Models 2
IBM 1443 Models
EMP NO
NAME
SALES AMOUNT
COMMISSION AMOUNT
CODE
0
1
2
3
4
5
6
7
8

```
   XX/XX/XX                        SALES COMMISSION REPORT                     PAGE XX0X

 EMP                               SALES                          COMMISSION
 NO              NAME              AMOUNT                           AMOUNT     CODE  STATUS

 XXX       XXXXXXXXXXXXXXXXXX      X,XX0.XX                            .XX       1   PART-TIME
 XXX       XXXXXXXXXXXXXXXXXX      X,XX0.XX                       X,XX0.XX       2   FULL-TIME
 XXX       XXXXXXXXXXXXXXXXXX      X,XX0.XX                       X,XX0.XX       3   COMMISSION

                 TOTAL SALES     $XX,XX0.XX  TOTAL COMMISSIONS   $XX,XX0.XX
```

```
    09/26/72                       SALES  COMMISSION  REPORT                PAGE   10

  EMP                              SALES                         COMMISSION
  NO              NAME             AMOUNT                          AMOUNT       CODE    STATUS

  115       ANDER, ARNOLD           195.00                            .00         1      PART-TIME
  192       BAKER, ROGER          5,000.00                         250.00         2      FULL-TIME
  237       LOUIE, JAMES          1,200.00                          64.00         3      COMMISSION
                                     .                               .
                                     .                               .
                                     .                               .
                 TOTAL SALES     $37,000.00     TOTAL COMMISSIONS  $ 2,530.00
```

TEST DATA – ASSIGNMENT NO. 2

Salesman Number (col 1-3)	Name (col 4-20)	Sales Amount (col 21-26)	Code (col 80)
115	Ander, Arnold	019500	1
192	Baker, Roger	500000	2
237	Louie, James	120000	3
376	Menton, Robert	326500	3
657	Operit, Fred	099500	3
961	Talbott, Mike	539000	2

CHAPTER 9

MINOR TOTAL LOGIC

In programming it is frequently desirable to be able to perform alternative functions depending upon conditions existing within the logic of the program. This concept of executing a sequence of instructions depending upon program logic may be accomplished by means of PROGRAMMED SWITCHES. For example, in the following illustration calculations are to be performed only if SWITCH 1 is ON. If SWITCH 1 is OFF, calculations are to be bypassed.

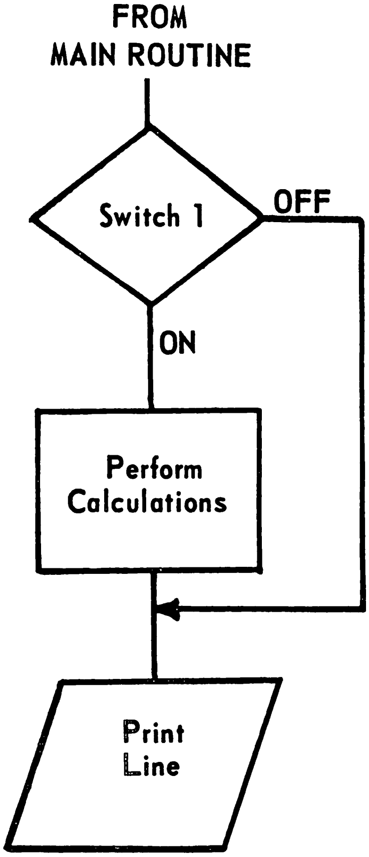


Figure 9-1 Programmed Switch

It should be noted that a Programmed Switch is not a physical switch on the console of the computer, but is merely a programming technique that allows alternative operations or sequences of instructions to be executed depending upon the status of the Switch.

The sample problem developed in this chapter illustrates an application in which a programmed switch is used. The input and output are illustrated in Figure 9-2. In this problem, an Accounts Receivable Register is to be prepared from Accounts Receivable Cards. A minor total is to be taken when there is a change in Customer Number and a final total is to be taken after all cards have been processed.

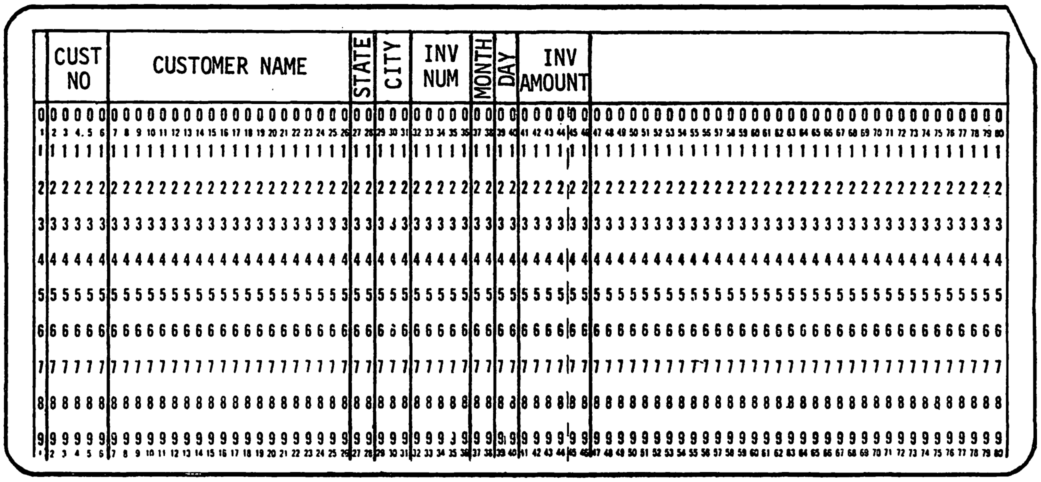
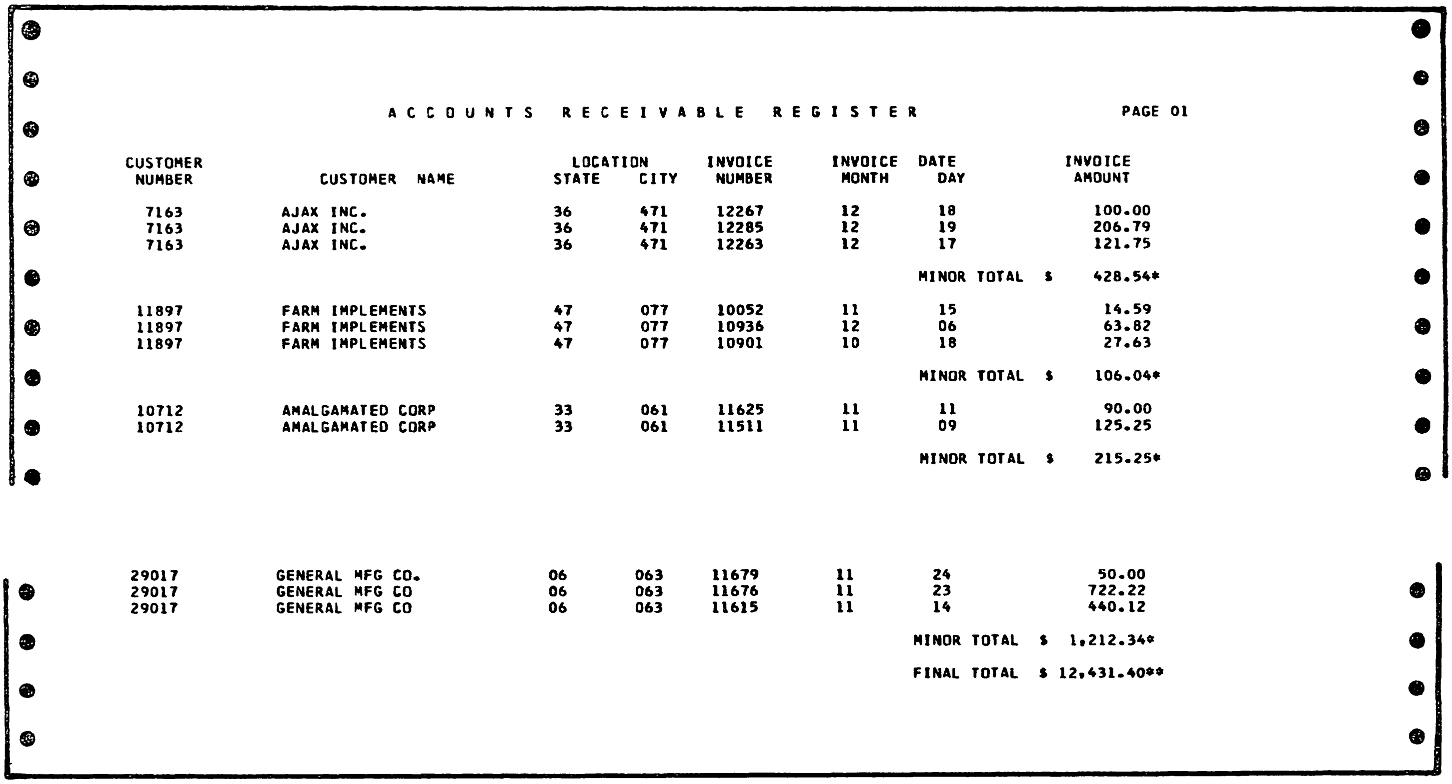


Figure 9-2 Input



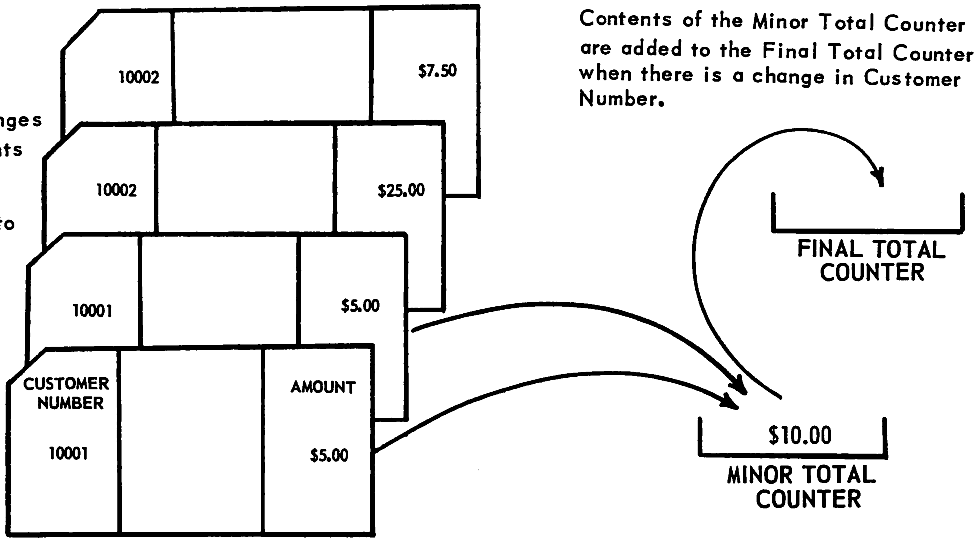
```
                    A C C O U N T S   R E C E I V A B L E   R E G I S T E R                 PAGE 01

CUSTOMER                                     LOCATION     INVOICE       INVOICE  DATE          INVOICE
 NUMBER           CUSTOMER  NAME            STATE   CITY   NUMBER        MONTH    DAY           AMOUNT

  7163         AJAX INC.                     36     471    12267          12       18            100.00
  7163         AJAX INC.                     36     471    12285          12       19            206.79
  7163         AJAX INC.                     36     471    12263          12       17            121.75

                                                                                 MINOR TOTAL  $   428.54*

 11897         FARM IMPLEMENTS               47     077    10052          11       15             14.59
 11897         FARM IMPLEMENTS               47     077    10936          12       06             63.82
 11897         FARM IMPLEMENTS               47     077    10901          10       18             27.63

                                                                                 MINOR TOTAL  $   106.04*

 10712         AMALGAMATED CORP              33     061    11625          11       11             90.00
 10712         AMALGAMATED CORP              33     061    11511          11       09            125.25

                                                                                 MINOR TOTAL  $   215.25*




 29017         GENERAL MFG CO.               06     063    11679          11       24             50.00
 29017         GENERAL MFG CO                06     063    11676          11       23            722.22
 29017         GENERAL MFG CO                06     063    11615          11       14            440.12

                                                                                 MINOR TOTAL  $ 1,212.34*

                                                                                 FINAL TOTAL  $ 12,431.40**
```

Figure 9-3 Output

In the sample problem, the logic required for the taking of a minor total is based upon the comparing of the Customer Numbers punched in the input data cards. If the Customer Numbers are the same, the Amount Field on the card is added to a minor total "counter" and the card is printed. When a card is read that contains a Customer Number that is different than the previous card, the contents of the Minor Total Counter are added to a Final Total Counter, the Minor Total Counter is moved to the output area, printed, and then cleared to zero. The counter, after being set to zero, can now be used to accumulate the total for the next series of items.

After all cards have been processed the contents of the final total counter are printed.

The diagram below illustrates the process of adding data from each individual card into a minor total counter, and then adding the minor total counter into the final total counter. Note that when using this technique the final total is the sum of the minor totals.



Contents of the Minor Total Counter are added to the Final Total Counter when there is a change in Customer Number.

When the Customer Number changes from 10001 to 10002, the contents of the Minor Total Counter are printed, and the contents of the Minor Total Counter are added to the Final Total Counter.

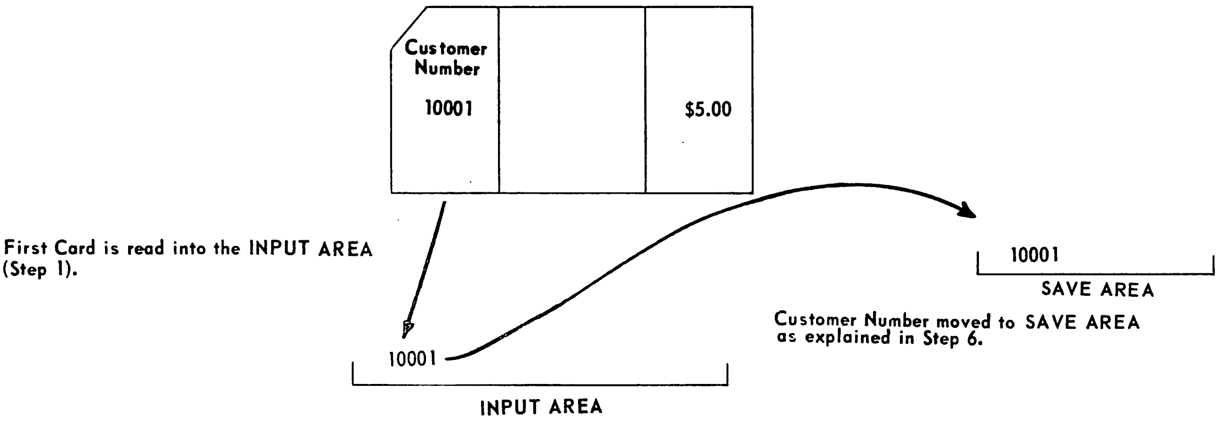
Amount for Customer 10001 is added to the Minor Total Counter and the card is printed.

Figure 9-4 Schematic Illustrating Adding to a Minor Total Counter and a Final Total Counter

The logic for processing the first card is explained below and is illustrated in Figure 9-6.

1. The first card is read.
2. Computer checks to see if it is the last card.
3. The Customer Number from the first card is compared to a "SAVE-AREA". The Save-Area is an area in WORKING-STORAGE that will be set to zero at the beginning of the execution of the program by a VALUE Clause; therefore, an unequal condition will exist when the first Customer Card is read and compared.
4. When the Customer Number is unequal, a branch occurs to the "040-SWITCH1-ROUTINE". The "programmed switch" at the 040-SWITCH1-ROUTINE is OFF at the beginning of the program; therefore, a branch will occur to the routine to turn the Switch ON.
5. The Switch is turned ON.
6. The Customer Number from the first card is moved to the SAVE-AREA.
7. A branch occurs to the routine to move the fields to be printed to the Print Area.
8. The Amount Field is added from the card into the Minor Total Counter.
9. The routine is executed to print the first card. A branch then occurs to read another card.

The contents of storage during execution of this portion of the program may be illustrated as follows.


Customer Number
10001
$5.00
First Card is read into the INPUT AREA (Step 1).
10001
INPUT AREA
10001
SAVE AREA
Customer Number moved to SAVE AREA as explained in Step 6.

Note that the Customer Number from the first card is now stored in both the INPUT AREA and the SAVE AREA.

Figure 9-5 Schematic Illustrating the Movement of a Control Field to a "Save Area"

When the second card is read into the system, for example, another card containing Customer Number 10001, an equal condition will occur when the compare instruction is executed. It can be seen from the flowchart that the Amount Field from this card will be added to the minor total counter, the card will be printed and a branch will occur to read another card. This sequence of instructions will be executed as long as the Customers are equal.

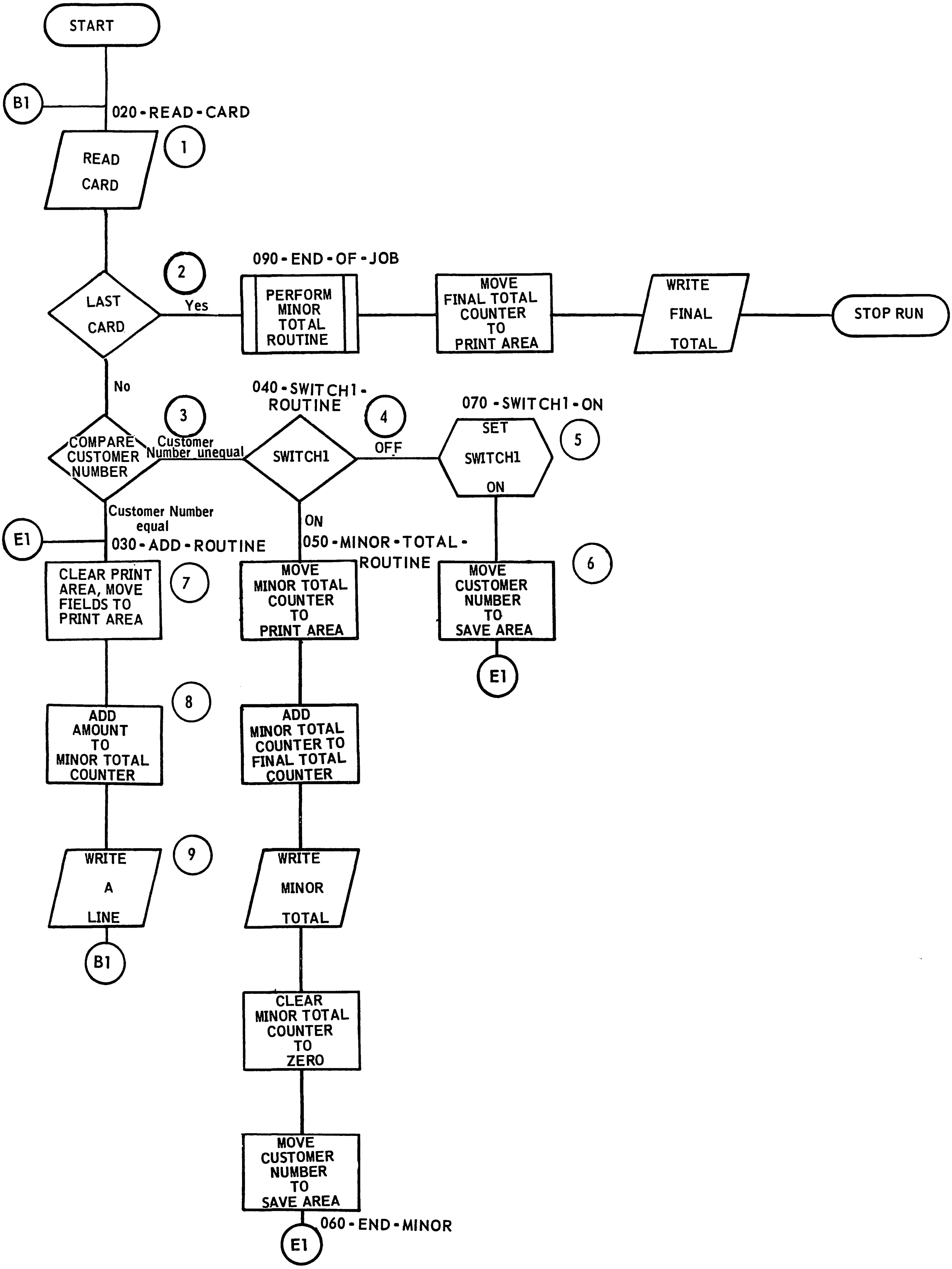
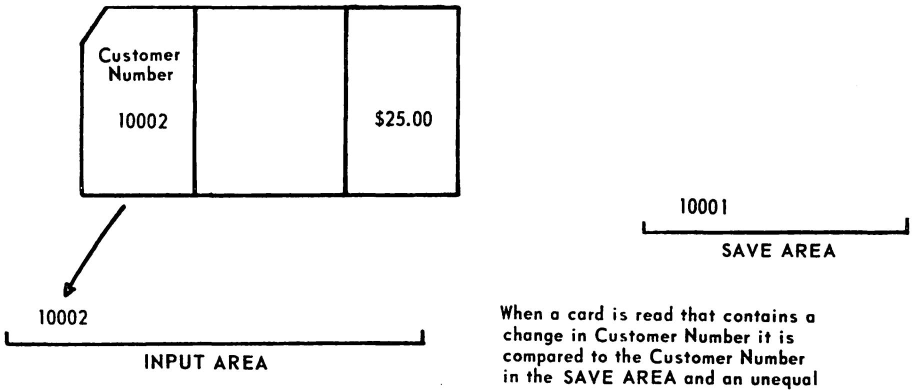

START
B1
020-READ-CARD
READ CARD
1
LAST CARD
2
Yes
090-END-OF-JOB
PERFORM MINOR TOTAL ROUTINE
MOVE FINAL TOTAL COUNTER TO PRINT AREA
WRITE FINAL TOTAL
STOP RUN
No
COMPARE CUSTOMER NUMBER
3
Customer Number unequal
040-SWITCH1-ROUTINE
SWITCH1
4
OFF
070-SWITCH1-ON
SET SWITCH1 ON
5
Customer Number equal
E1
030-ADD-ROUTINE
ON
050-MINOR-TOTAL-ROUTINE
CLEAR PRINT AREA, MOVE FIELDS TO PRINT AREA
7
MOVE MINOR TOTAL COUNTER TO PRINT AREA
MOVE CUSTOMER NUMBER TO SAVE AREA
6
E1
ADD AMOUNT TO MINOR TOTAL COUNTER
8
ADD MINOR TOTAL COUNTER TO FINAL TOTAL COUNTER
WRITE A LINE
9
B1
WRITE MINOR TOTAL
CLEAR MINOR TOTAL COUNTER TO ZERO
MOVE CUSTOMER NUMBER TO SAVE AREA
060-END-MINOR
E1

Figure 9-6 Flowchart illustrating the processing of the first card

When a card is read that contains a Customer Number that is unequal to the previous card, the following sequence of instructions are executed. The related flowchart is illustrated on the following page.

1. A card is read that contains a Customer Number that is different from the previous card.
2. If it is not the last card, the Compare Statement is executed.
3. The Customer Number Fields are compared and an unequal condition exists. A branch occurs to the 040-SWITCH1-ROUTINE.
4. As the "programmed switch" is now ON (it was turned ON when the first card was processed), the 050-MINOR-TOTAL-ROUTINE is executed.
5. The Minor Total Counter is moved to the Print Area.
6. The Minor Total Counter is added to the Final Total Counter.
7. The Minor Total Counter is printed.
8. The Minor Total Counter is cleared to zero.
9. The Customer Number from the card that caused the unequal condition will be moved to the SAVE-AREA and a branch occurs to the 030-ADD-ROUTINE where the fields on the card are moved to the Print Area, the Amount Field from the card that caused the "control change" is added to the Minor Total Counter and a detail line is printed. A branch then occurs to read another card. The logic illustrated applies to all other cards in the file being processed.
10. When the last card is processed, the statement to perform the 050-MINOR-TOTAL-ROUTINE is executed (this causes the Minor Total for the last group of cards to be printed). After the 050-MINOR-TOTAL-ROUTINE is executed, the instruction following the PERFORM statement is executed next.
11. The Final Total is moved to the Output Area and Printed and the execution of the problem is completed.

The following diagrams illustrate the contents of storage during execution of this segment of the program.



When a card is read that contains a change in Customer Number it is compared to the Customer Number in the SAVE AREA and an unequal comparsion results.

Figure 9-7 Schematic illustration of new Control Number moved to "Save Area"

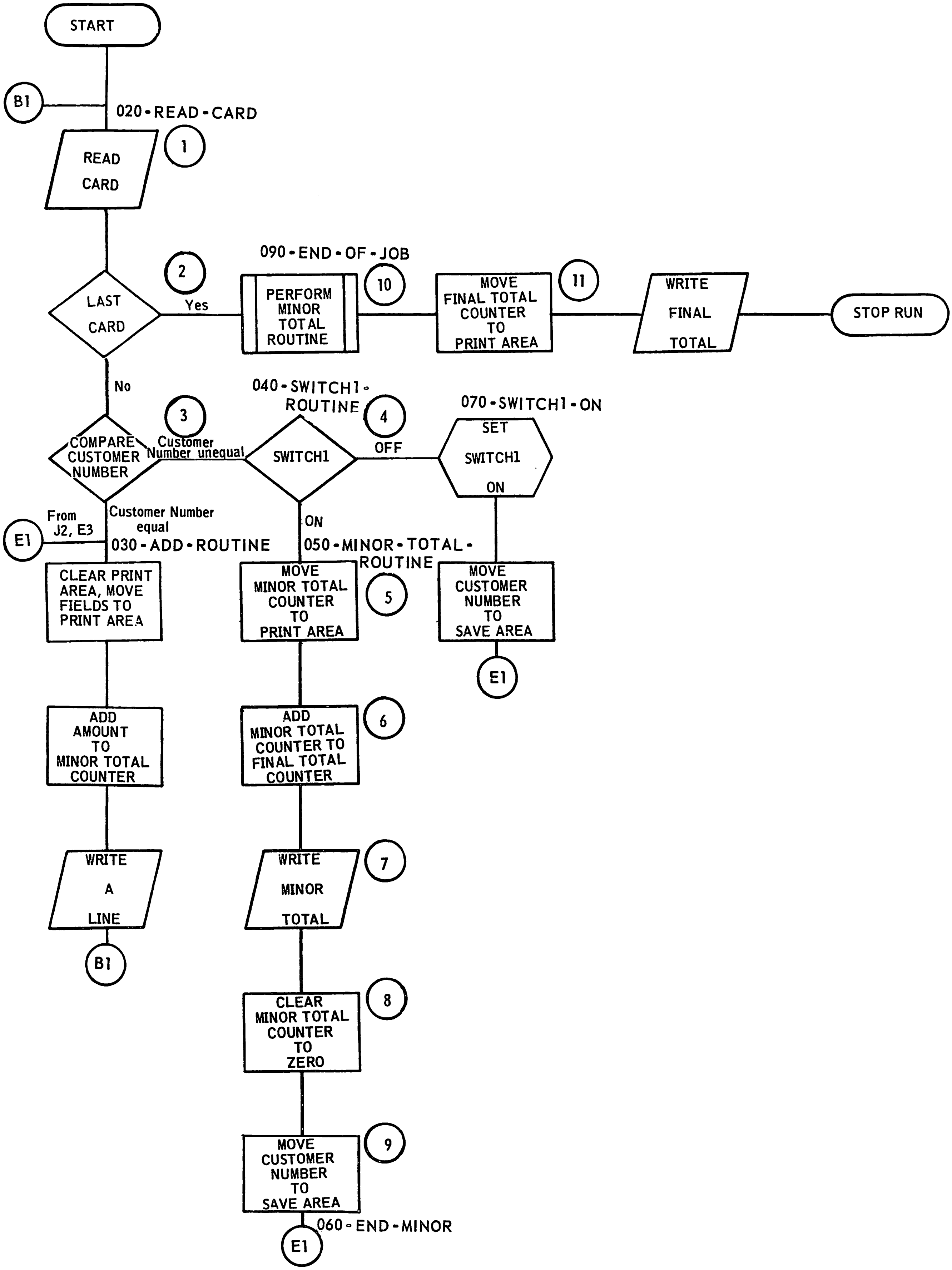
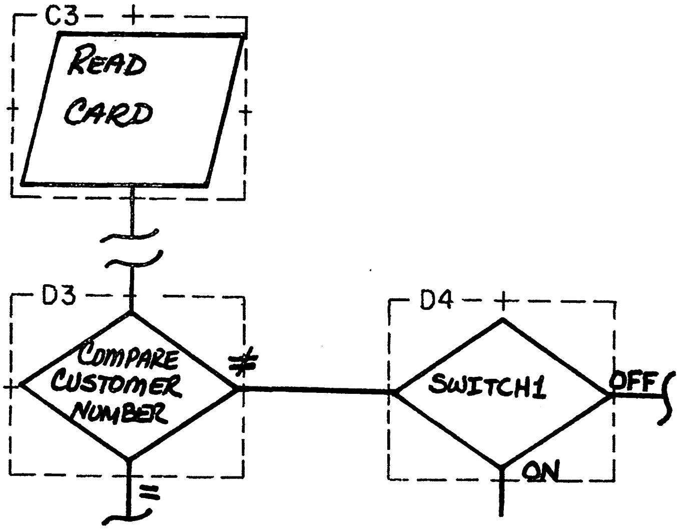

START
B1
020-READ-CARD
READ CARD
1
LAST CARD
2
Yes
No
090-END-OF-JOB
PERFORM MINOR TOTAL ROUTINE
10
MOVE FINAL TOTAL COUNTER TO PRINT AREA
11
WRITE FINAL TOTAL
STOP RUN
COMPARE CUSTOMER NUMBER
3
Customer Number unequal
040-SWITCH1-ROUTINE
4
SWITCH1
OFF
070-SWITCH1-ON
SET SWITCH1 ON
From J2, E3
E1
Customer Number equal
030-ADD-ROUTINE
CLEAR PRINT AREA, MOVE FIELDS TO PRINT AREA
ADD AMOUNT TO MINOR TOTAL COUNTER
WRITE A LINE
B1
ON
050-MINOR-TOTAL-ROUTINE
MOVE MINOR TOTAL COUNTER TO PRINT AREA
5
ADD MINOR TOTAL COUNTER TO FINAL TOTAL COUNTER
6
WRITE MINOR TOTAL
7
CLEAR MINOR TOTAL COUNTER TO ZERO
8
MOVE CUSTOMER NUMBER TO SAVE AREA
9
060-END-MINOR
E1
MOVE CUSTOMER NUMBER TO SAVE AREA
E1

Figure 9-8 Flowchart illustrating processing when unequal control numbers are detected

It should again be noted that a "Programmed Switch" is merely a programming technique which allows alternative operations to be performed depending upon the status of the switch. A variety of techniques are available to the COBOL programmer. The technique illustrated in the sample problem is to use a conditional test (IF Statement).

In the sample program, a Programmed Switch is used when the first card is processed to provide a means of storing the Customer Number for the first card in a "Save Area", so that the first card may be compared to subsequent cards when they are read. A segment of the flowchart and the related coding are illustrated below.



```
WORKING-STORAGE SECTION.
77  SWITCH1                 PICTURE XXX  VALUE  'OFF'.
```

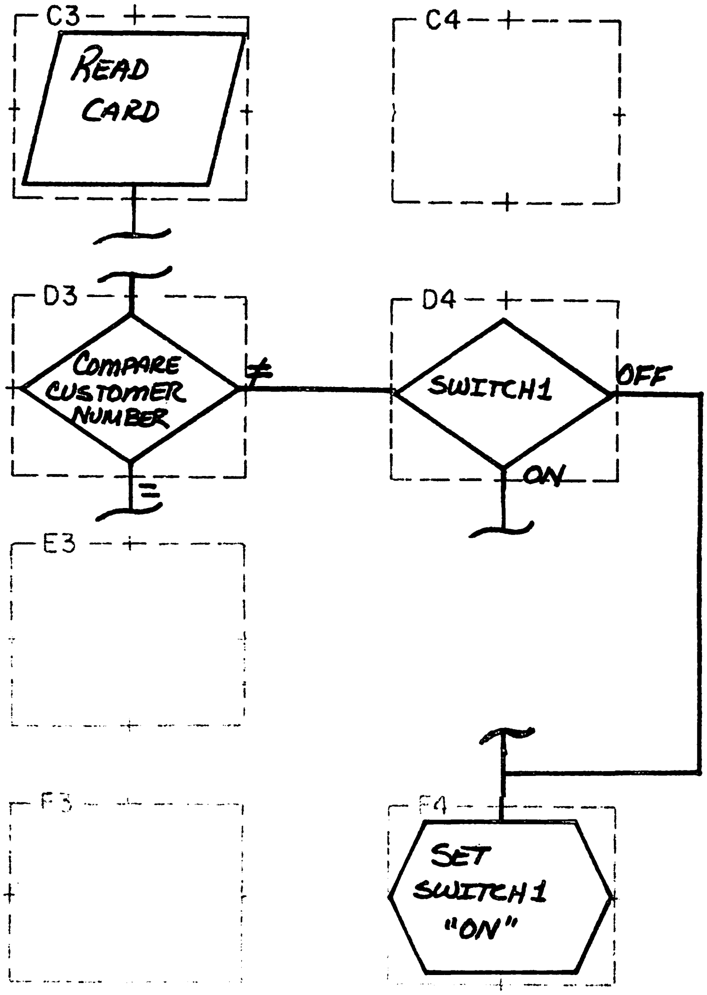
```
040-SWITCH1-ROUTINE.
    IF SWITCH1 = 'OFF' GO TO 070-SWITCH1-ON.
```

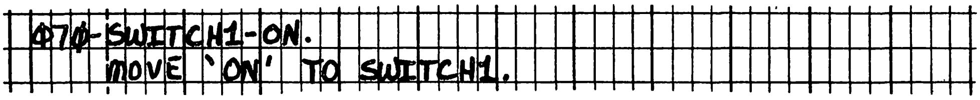
Figure 9-9 Flowchart and Coding for a Programmed Switch

In the example above, an area of storage is reserved in WORKING-STORAGE called SWITCH1 with a value of "OFF". In the Procedure Division the contents of SWITCH1 can be tested by means of an IF statement. Thus, the switch portion of the flowchart is nothing more than an IF statement.

The following segment of a flowchart and related coding illustrates the programming technique used to turn a Programmed Switch ON.

To turn the Switch "ON" the literal "ON" is moved into the area referenced by SWITCH1. The following segment of the flowchart illustrates the COBOL Statement to turn SWITCH1 "ON".


C3
READ CARD
C4
D3
COMPARE CUSTOMER NUMBER
≠
=
D4
SWITCH1
OFF
ON
E3
F3
F4
SET SWITCH1 "ON"



```
070-SWITCH1-ON.
    MOVE 'ON' TO SWITCH1.
```

Figure 9-10 Flowchart and Coding for turning a Programmed Switch "ON"

It should be noted that any values could have been established in WORKING-STORAGE for SWITCH1. The words "ON" and "OFF" were selected because they are representative of the operation being performed.

SAMPLE PROBLEM

The complete documentation for the sample problem is illustrated below and on the following pages.

INPUT

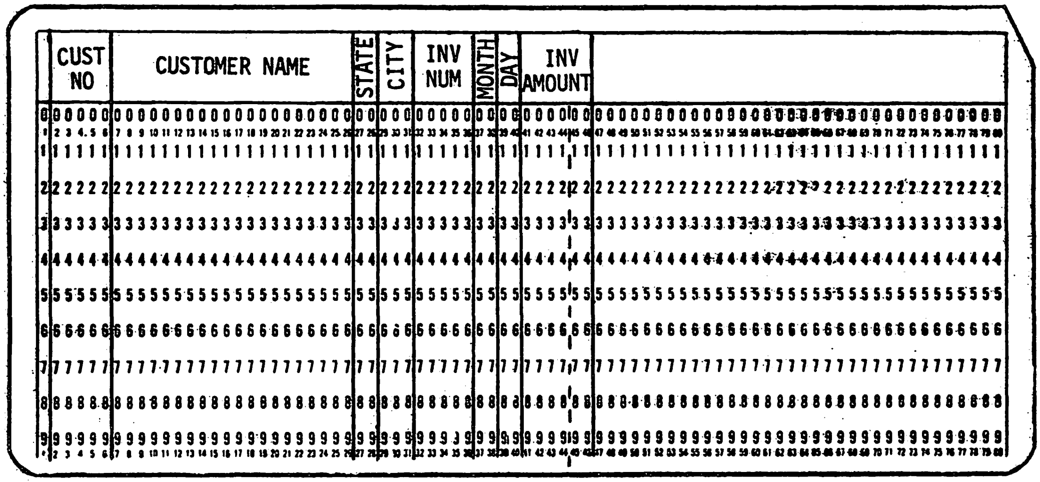
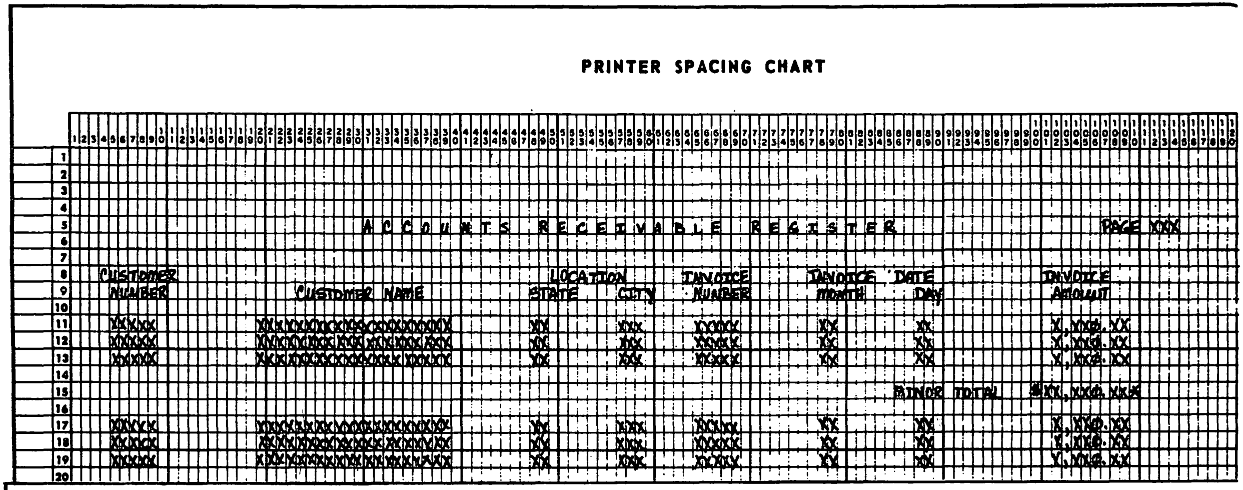
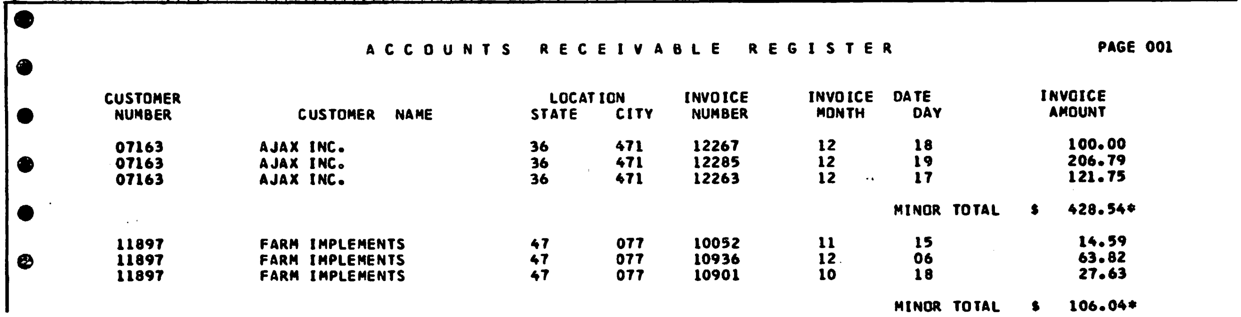

CUST NO
CUSTOMER NAME
STATE
CITY
INV NUM
MONTH
DAY
INV AMOUNT

Figure 9-11 Input

OUTPUT


PRINTER SPACING CHART
ACCOUNTS RECEIVABLE REGISTER
PAGE XXX
CUSTOMER NUMBER
CUSTOMER NAME
LOCATION STATE CITY
INVOICE NUMBER
INVOICE MONTH
DATE DAY
INVOICE AMOUNT
XXXXX
XX
XXX
X,XXX.XX
MINOR TOTAL
$XX,XXX.XX*



```
                         A C C O U N T S   R E C E I V A B L E   R E G I S T E R                     PAGE 001

CUSTOMER                                             LOCATION      INVOICE     INVOICE  DATE            INVOICE
 NUMBER              CUSTOMER  NAME               STATE    CITY    NUMBER       MONTH    DAY             AMOUNT

 07163        AJAX INC.                            36       471     12267        12       18              100.00
 07163        AJAX INC.                            36       471     12285        12       19              206.79
 07163        AJAX INC.                            36       471     12263        12       17              121.75

                                                                                    MINOR TOTAL    $     428.54*

 11897        FARM IMPLEMENTS                      47       077     10052        11       15               14.59
 11897        FARM IMPLEMENTS                      47       077     10936        12       06               63.82
 11897        FARM IMPLEMENTS                      47       077     10901        10       18               27.63

                                                                                    MINOR TOTAL    $     106.04*
```

Figure 9-12 Output

Flowchart

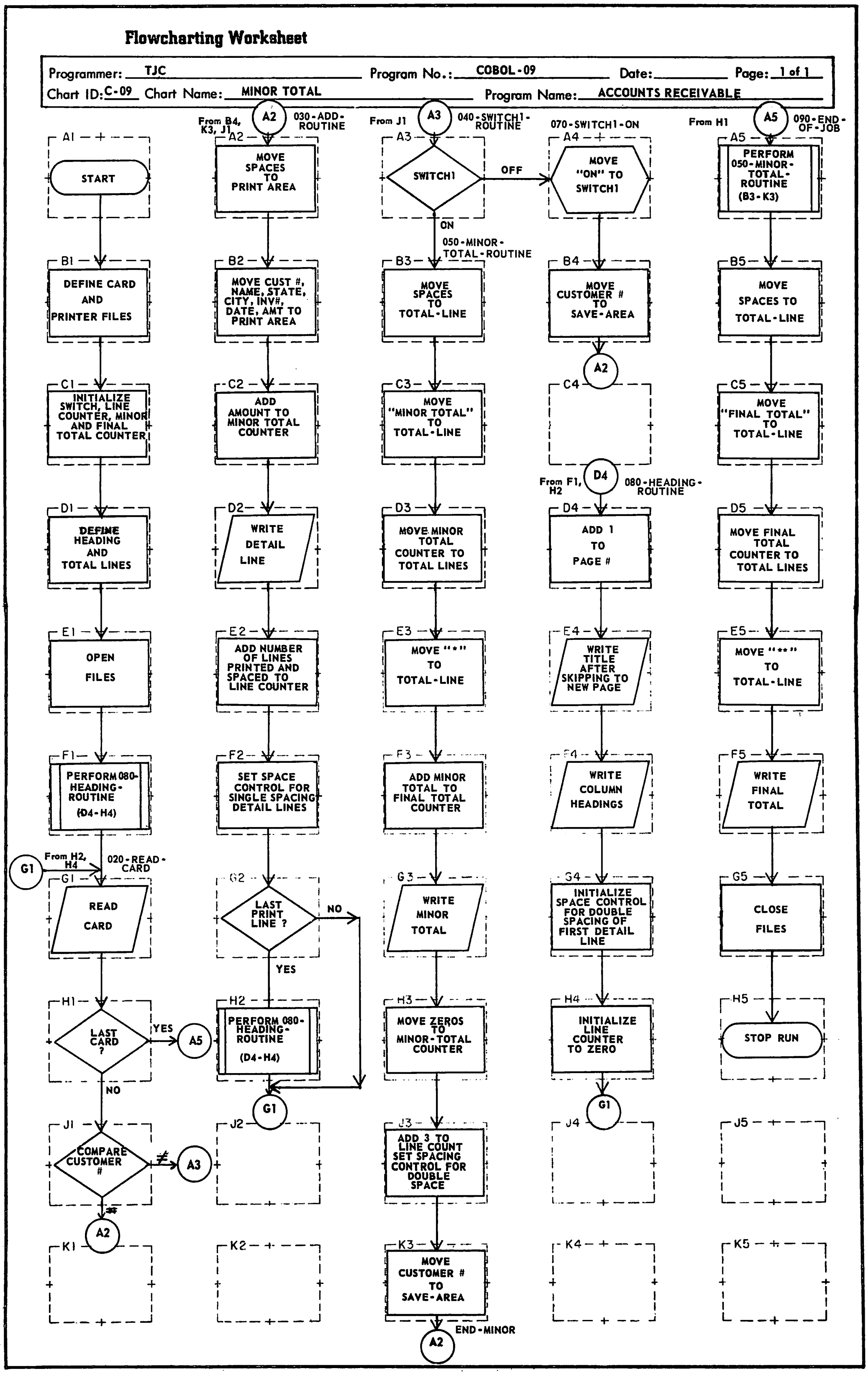

Flowcharting Worksheet
Programmer: TJC
Program No.: COBOL-09
Date:
Page: 1 of 1
Chart ID: C-09
Chart Name: MINOR TOTAL
Program Name: ACCOUNTS RECEIVABLE
A1
START
B1
DEFINE CARD AND PRINTER FILES
C1
INITIALIZE SWITCH, LINE COUNTER, MINOR AND FINAL TOTAL COUNTER
D1
~~DEFINE~~ HEADING AND TOTAL LINES
E1
OPEN FILES
F1
PERFORM 080-HEADING-ROUTINE (D4-H4)
G1
From H2, H4
020-READ-CARD
READ CARD
H1
LAST CARD ?
YES
A5
NO
J1
COMPARE CUSTOMER #
≠
A3
=
A2
K1
From B4, K3, J1
A2
030-ADD-ROUTINE
MOVE SPACES TO PRINT AREA
B2
MOVE CUST #, NAME, STATE, CITY, INV#, DATE, AMT TO PRINT AREA
C2
ADD AMOUNT TO MINOR TOTAL COUNTER
D2
WRITE DETAIL LINE
E2
ADD NUMBER OF LINES PRINTED AND SPACED TO LINE COUNTER
F2
SET SPACE CONTROL FOR SINGLE SPACING DETAIL LINES
G2
LAST PRINT LINE ?
NO
YES
H2
PERFORM 080-HEADING-ROUTINE (D4-H4)
G1
J2
K2
From J1
A3
040-SWITCH1-ROUTINE
SWITCH1
OFF
ON
050-MINOR-TOTAL-ROUTINE
B3
MOVE SPACES TO TOTAL-LINE
C3
MOVE "MINOR TOTAL" TO TOTAL-LINE
D3
MOVE MINOR TOTAL COUNTER TO TOTAL LINES
E3
MOVE "*" TO TOTAL-LINE
F3
ADD MINOR TOTAL TO FINAL TOTAL COUNTER
G3
WRITE MINOR TOTAL
H3
MOVE ZEROS TO MINOR-TOTAL COUNTER
J3
ADD 3 TO LINE COUNT SET SPACING CONTROL FOR DOUBLE SPACE
K3
MOVE CUSTOMER # TO SAVE-AREA
END-MINOR
A2
070-SWITCH1-ON
A4
MOVE "ON" TO SWITCH1
B4
MOVE CUSTOMER # TO SAVE-AREA
A2
C4
From F1, H2
D4
080-HEADING-ROUTINE
ADD 1 TO PAGE #
E4
WRITE TITLE AFTER SKIPPING TO NEW PAGE
F4
WRITE COLUMN HEADINGS
G4
INITIALIZE SPACE CONTROL FOR DOUBLE SPACING OF FIRST DETAIL LINE
H4
INITIALIZE LINE COUNTER TO ZERO
G1
J4
K4
From H1
A5
090-END-OF-JOB
PERFORM 050-MINOR-TOTAL-ROUTINE (B3-K3)
B5
MOVE SPACES TO TOTAL-LINE
C5
MOVE "FINAL TOTAL" TO TOTAL-LINE
D5
MOVE FINAL TOTAL COUNTER TO TOTAL LINES
E5
MOVE "**" TO TOTAL-LINE
F5
WRITE FINAL TOTAL
G5
CLOSE FILES
H5
STOP RUN
J5
K5

Figure 9-13 Flowchart

SAMPLE PROGRAM

```
     1                                  IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4             03/07/73

00001   001010 IDENTIFICATION DIVISION.                                         MINTOTAL
00002   001020 PROGRAM-ID. PROB09.                                              MINTOTAL
00003   001030 AUTHOR. TJC.                                                     MINTOTAL
00004   001040 INSTALLATION. LONG BEACH.                                        MINTOTAL
00005   001050 DATE-WRITTEN. 03/21/73.                                          MINTOTAL
00006   001060 DATE-COMPILED. 03/22/73.                                         MINTOTAL
00007   001070 SECURITY. UNCLASSIFIED.                                          MINTOTAL
00008   001080 REMARKS. THIS PROGRAM PRODUCES AN ACCOUNTS RECEIVABLE            MINTOTAL
00009   001090     REGISTER AND TAKES A MINOR TOTAL WHEN THERE IS A CHANGE IN   MINTOTAL
00010   001100     CUSTOMER NUMBER.                                             MINTOTAL
00011   001110 ENVIRONMENT DIVISION.                                            MINTOTAL
00012   001120 CONFIGURATION SECTION.                                           MINTOTAL
00013   001130 SOURCE-COMPUTER. IBM-360-F30.                                    MINTOTAL
00014   001140 OBJECT-COMPUTER. IBM-360-F30.                                    MINTOTAL
00015   001150 SPECIAL-NAMES.                                                   MINTOTAL
00016   001160     C01 IS TO-TOP-OF-PAGE.                                       MINTOTAL
00017   001170 INPUT-OUTPUT SECTION.                                            MINTOTAL
00018   001180 FILE-CONTROL.                                                    MINTOTAL
00019   001190     SELECT ACCOUNTS-RECEIVABLE-FILE                              MINTOTAL
00020   001210         ASSIGN TO SYS007-UR-2540R-S.                             MINTOTAL
00021   002020     SELECT PRINT-FILE                                            MINTOTAL
00022   002030         ASSIGN TO SYS008-UR-1403-S,                              MINTOTAL
00023   002040 DATA DIVISION.                                                   MINTOTAL
00024   002050 FILE SECTION.                                                    MINTOTAL
00025   002060 FD  ACCOUNTS-RECEIVABLE-FILE,                                    MINTOTAL
00026   002080     RECORD CONTAINS 80 CHARACTERS,                               MINTOTAL
00027   002090     LABEL RECORDS ARE OMITTED,                                   MINTOTAL
00028   002100     DATA RECORD IS CARD.                                         MINTOTAL
00029   002110 01  CARD                        PICTURE X(80).                   MINTOTAL
00030   002120 FD  PRINT-FILE,                                                  MINTOTAL
00031   002140     RECORD CONTAINS 133 CHARACTERS,                              MINTOTAL
00032   002150     LABEL RECORDS ARE OMITTED,                                   MINTOTAL
00033   002160     DATA RECORD IS LINE-OUT.                                     MINTOTAL
00034   002170 01  LINE-OUT                    PICTURE X(133).                  MINTOTAL
00035   002180 WORKING-STORAGE SECTION.                                         MINTOTAL
00036   002190 77  SWITCH1                     PICTURE XXX VALUE 'OFF'.         MINTOTAL
00037   002200 77  VARIABLE                    PICTURE 9.                       MINTOTAL
00038   003010 77  SAVE-AREA                   PICTURE X(5) VALUE ZERO.         MINTOTAL
00039   003020 77  MINOR-TOTAL                 PICTURE S9(5)V99 VALUE ZERO      MINTOTAL
00040   003030                                     COMPUTATIONAL-3.             MINTOTAL
00041   003040 77  FINAL-TOTAL                 PICTURE S9(5)V99 VALUE ZERO      MINTOTAL
00042   003050                                     COMPUTATIONAL-3.             MINTOTAL
00043   003052 77  LINE-COUNT                  PICTURE S999 VALUE ZERO          MINTOTAL
00044   003053                                     COMPUTATIONAL-3.             MINTOTAL
00045   003060 01  AR-CARD.                                                     MINTOTAL
00046   003070     03  FILLER                  PICTURE X.                       MINTOTAL
00047   003080     03  CUST-NO-CD              PICTURE X(5).                    MINTOTAL
00048   003090     03  CUST-NAME-CD            PICTURE X(20).                   MINTOTAL
00049   003100     03  STATE-CD                PICTURE XX.                      MINTOTAL
00050   003110     03  CITY-CD                 PICTURE XXX.                     MINTOTAL
00051   003120     03  INV-NUM-CD              PICTURE X(5).                    MINTOTAL
00052   003130     03  MONTH-CD                PICTURE XX.                      MINTOTAL
00053   003140     03  DAY-CD                  PICTURE XX.                      MINTOTAL
00054   003150     03  INV-AMOUNT-CD           PICTURE 9999V99.                 MINTOTAL
00055   003160     03  FILLER                  PICTURE X(34).                   MINTOTAL
00056   003170 01  DETAIL-LINE.                                                 MINTOTAL
00057   003180     03  FILLER                  PICTURE X.                       MINTOTAL
00058   003190     03  FILLER                  PICTURE X(4).                    MINTOTAL
00059   003200     03  CUST-NO-PR              PICTURE X(5).                    MINTOTAL
00060   004010     03  FILLER                  PICTURE X(10).                   MINTOTAL
00061   004020     03  CUST-NAME-PR            PICTURE X(20).                   MINTOTAL
00062   004030     03  FILLER                  PICTURE X(8).                    MINTOTAL
00063   004040     03  STATE-PR                PICTURE XX.                      MINTOTAL
00064   004050     03  FILLER                  PICTURE X(7).                    MINTOTAL
00065   004060     03  CITY-PR                 PICTURE XXX.                     MINTOTAL
00066   004070     03  FILLER                  PICTURE X(5).                    MINTOTAL
00067   004080     03  INV-NUM-PR              PICTURE X(5).                    MINTOTAL
00068   004090     03  FILLER                  PICTURE X(8).                    MINTOTAL
00069   004100     03  MONTH-PR                PICTURE XX.                      MINTOTAL
00070   004110     03  FILLER                  PICTURE X(8).                    MINTOTAL
00071   004120     03  DAY-PR                  PICTURE XX.                      MINTOTAL
00072   004130     03  FILLER                  PICTURE X(12).                   MINTOTAL
00073   004140     03  INV-AMOUNT-PR           PICTURE Z,ZZZ.99.                MINTOTAL
00074   004150     03  FILLER                  PICTURE X(23).                   MINTOTAL
00075   004160 01  HEAD1.                                                       MINTOTAL
00076   004170     03  FILLER                  PICTURE X.                       MINTOTAL
00077   004180     03  FILLER                  PICTURE X(30) VALUE SPACES.      MINTOTAL
00078   004190     03  FILLER                  PICTURE X(76) VALUE              MINTOTAL
00079   004200      'A C C O U N T S   R E C E I V A B L E   R E G I S T E R'.  MINTOTAL
00080   005010     03  FILLER                  PICTURE X(5) VALUE 'PAGE '.      MINTOTAL
00081   005020     03  PAGE-NO                 PICTURE 999 VALUE ZERO.          MINTOTAL
00082   005030     03  FILLER                  PICTURE X(18) VALUE SPACES.      MINTOTAL
00083   005040 01  HEAD2.                                                       MINTOTAL
00084   005050     03  FILLER                  PICTURE X.                       MINTOTAL
00085   005060     03  FILLER                  PICTURE X(11) VALUE              MINTOTAL
00086   005070       '   CUSTOMER'.                                             MINTOTAL
00087   005080     03  FILLER                  PICTURE X(38) VALUE SPACES.      MINTOTAL
```

```
00088   005090     03  FILLER                 PICTURE X(40) VALUE             MINTOTAL
00089   005100       'LOCATION      INVOICE      INVOICE  DATE'.              MINTOTAL
00090   005110     03  FILLER                 PICTURE X(11) VALUE SPACES.     MINTOTAL
00091   005120     03  FILLER                 PICTURE X(7) VALUE 'INVOICE'.   MINTOTAL
00092   005130     03  FILLER                 PICTURE X(25) VALUE SPACES.     MINTOTAL
00093   005140 01  HEAD3.                                                     MINTOTAL
00094   005150     03  FILLER                 PICTURE X.                      MINTOTAL
00095   005160     03  FILLER                 PICTURE X(20) VALUE             MINTOTAL
00096   005170      '    NUMBER'.                                             MINTOTAL
00097   005180     03  FILLER                 PICTURE X(13) VALUE SPACES.     MINTOTAL
00098   005190     03  FILLER                 PICTURE X(47) VALUE             MINTOTAL
00099   005200      'CUSTOMER  NAME          STATE    CITY    NUMBER'.        MINTOTAL
00100   006010     03  FILLER                 PICTURE X(37) VALUE             MINTOTAL
00101   006020      '       MONTH     DAY           AMOUNT'.                  MINTOTAL
00102   006030     03  FILLER                 PICTURE X(25) VALUE SPACES.     MINTOTAL
00103   006040 01  TOTAL-LINE.                                                MINTOTAL
00104   006050     03  FILLER                 PICTURE X.                      MINTOTAL
00105   006060     03  FILLER                 PICTURE X(85).                  MINTOTAL
00106   006070     03  MSG                    PICTURE X(14).                  MINTOTAL
00107   006080     03  TOTAL-OUT              PICTURE $ZZ,ZZZ.99.             MINTOTAL
00108   006090     03  STAR                   PICTURE XX.                     MINTOTAL
00109   006100     03  FILLER                 PICTURE X(21) VALUE SPACES.     MINTOTAL
00110   006110 PROCEDURE DIVISION.                                            MINTOTAL
00111   006120 010-BEGIN-JOB.                                                 MINTOTAL
00112   006130     OPEN INPUT ACCOUNTS-RECEIVABLE-FILE, OUTPUT PRINT-FILE.    MINTOTAL
00113   006140     PERFORM 080-HEADING-ROUTINE THRU 080-EXIT.
00114   006150 020-READ-CARD.                                                 MINTOTAL
00115   006160     READ ACCOUNTS-RECEIVABLE-FILE INTO AR-CARD                 MINTOTAL
00116   006170         AT END GO TO 090-END-OF-JOB.                           MINTOTAL
00117   006180     IF CUST-NO-CD NOT = SAVE-AREA GO TO 040-SWITCH1-ROUTINE.   MINTOTAL
00118   006190 030-ADD-ROUTINE.                                               MINTOTAL
00119   006200     MOVE SPACES TO DETAIL-LINE.                                MINTOTAL
00120   007010     MOVE CUST-NO-CD TO CUST-NO-PR.                             MINTOTAL
00121   007020     MOVE CUST-NAME-CD TO CUST-NAME-PR.                         MINTOTAL
00122   007030     MOVE STATE-CD TO STATE-PR.                                 MINTOTAL
00123   007040     MOVE CITY-CD TO CITY-PR.                                   MINTOTAL
00124   007050     MOVE INV-NUM-CD TO INV-NUM-PR.                             MINTOTAL
00125   007060     MOVE MONTH-CD TO MONTH-PR.                                 MINTOTAL
00126   007070     MOVE DAY-CD TO DAY-PR.                                     MINTOTAL
00127   007080     MOVE INV-AMOUNT-CD TO INV-AMOUNT-PR.                       MINTOTAL
00128   007090     ADD INV-AMOUNT-CD TO MINOR-TOTAL.                          MINTOTAL
00129   007100     WRITE LINE-OUT FROM DETAIL-LINE AFTER ADVANCING            MINTOTAL
00130   007110         VARIABLE LINES.                                        MINTOTAL
00131   007130     MOVE '1' TO VARIABLE.                                      MINTOTAL
00132   007135     ADD 1 TO LINE-COUNT.                                       MINTOTAL
00133   007137     IF LINE-COUNT > 51 PERFORM 080-HEADING-ROUTINE THRU 080-EXIT.MINTOTAL
00134   007140     GO TO 020-READ-CARD.                                       MINTOTAL
00135   007150 040-SWITCH1-ROUTINE.                                           MINTOTAL
00136   007160     IF SWITCH1 = 'OFF' GO TO 070-SWITCH1-ON.                   MINTOTAL
00137   007170 050-MINOR-TOTAL-ROUTINE.                                       MINTOTAL
00138   007180     MOVE SPACES TO TOTAL-LINE.                                 MINTOTAL
00139   007190     MOVE 'MINOR TOTAL' TO MSG.                                 MINTOTAL
00140   007200     MOVE MINOR-TOTAL TO TOTAL-OUT.                             MINTOTAL
00141   008010     MOVE '*' TO STAR.                                          MINTOTAL
00142   008020     ADD MINOR-TOTAL TO FINAL-TOTAL.                            MINTOTAL
00143   008030     WRITE LINE-OUT FROM TOTAL-LINE AFTER ADVANCING 2 LINES.    MINTOTAL
00144   008050     MOVE ZERO TO MINOR-TOTAL.                                  MINTOTAL
00145   008055     ADD 3 TO LINE-COUNT.                                       MINTOTAL
00146   008060     MOVE '2' TO VARIABLE.                                      MINTOTAL
00147   008070     MOVE CUST-NO-CD TO SAVE-AREA.                              MINTOTAL
00148   008080 060-END-MINOR.                                                 MINTOTAL
00149   008090     GO TO 030-ADD-ROUTINE.                                     MINTOTAL
00150   008100 070-SWITCH1-ON.                                                MINTOTAL
00151   008110     MOVE 'ON' TO SWITCH1.                                      MINTOTAL
00152   008120     MOVE CUST-NO-CD TO SAVE-AREA.                              MINTOTAL
00153   008130     GO TO 030-ADD-ROUTINE.                                     MINTOTAL
00154   008140 080-HEADING-ROUTINE.                                           MINTOTAL
00155   008150     ADD 1 TO PAGE-NO.                                          MINTOTAL
00156   008160     WRITE LINE-OUT FROM HEAD1 AFTER ADVANCING TO-TOP-OF-PAGE.  MINTOTAL
00157   008170     WRITE LINE-OUT FROM HEAD2 AFTER ADVANCING 3 LINES.         MINTOTAL
00158   008180     WRITE LINE-OUT FROM HEAD3 AFTER ADVANCING 1 LINES.         MINTOTAL
00159   008185     MOVE ZERO TO LINE-COUNT.                                   MINTOTAL
00160   008190     MOVE '2' TO VARIABLE.                                      MINTOTAL
00161   008195 080-EXIT.  EXIT.                                               MINTOTAL
00162   008200 090-END-OF-JOB.                                                MINTOTAL
00163   009010     PERFORM 050-MINOR-TOTAL-ROUTINE.                           MINTOTAL
00164   009020     MOVE SPACES TO TOTAL-LINE.                                 MINTOTAL
00165   009030     MOVE 'FINAL TOTAL' TO MSG.                                 MINTOTAL
00166   009040     MOVE FINAL-TOTAL TO TOTAL-OUT.                             MINTOTAL
00167   009050     MOVE '**' TO STAR.                                         MINTOTAL
00168   009060     WRITE LINE-OUT FROM TOTAL-LINE AFTER ADVANCING 2 LINES.    MINTOTAL
00169   009070     CLOSE PRINT-FILE, ACCOUNTS-RECEIVABLE-FILE.                MINTOTAL
00170   009080     STOP RUN.                                                  MINTOTAL
```

CHAPTER 9

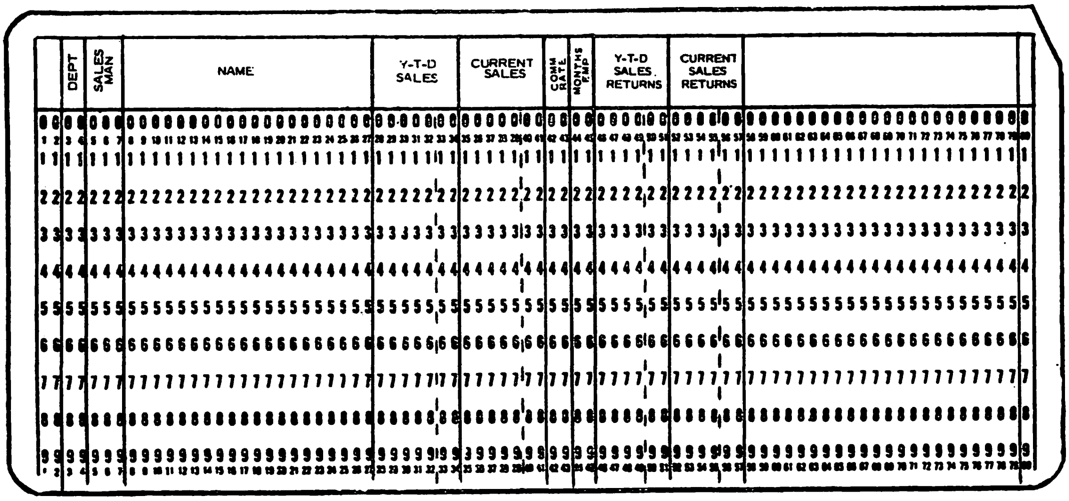
PROGRAMMING ASSIGNMENT 1

INSTRUCTIONS

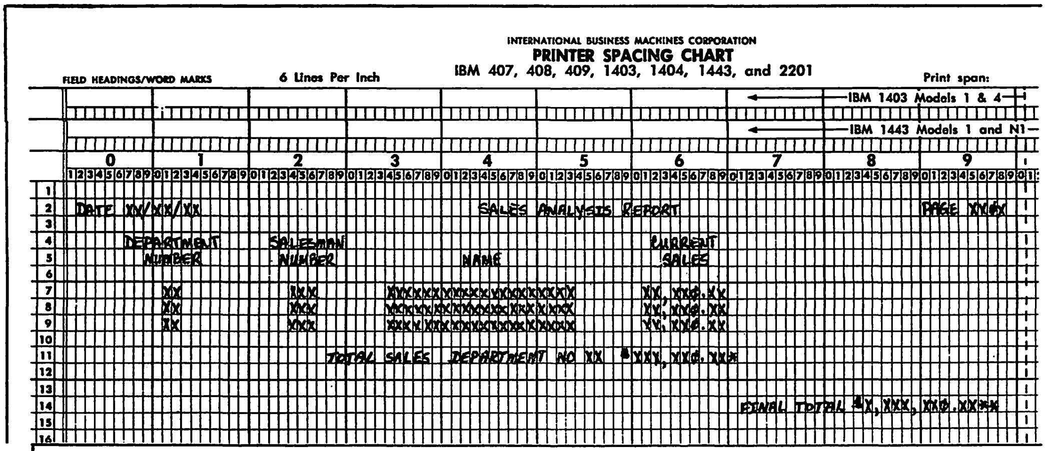
From the Sales cards illustrated below, a Sales Analysis Report is to be prepared. The report is to list the Department Number, Salesman Number, Name, and Current Sales. When there is a change in Department Number, a total for the department is to be taken. After all of the cards have been processed, a Final Total is to be printed.

INPUT: Sales Cards

The fields to be used in the Sales cards are the Department Number, the Salesman Number, the Name, and the Current Sales.



OUTPUT: Sales Analysis Report


INTERNATIONAL BUSINESS MACHINES CORPORATION

PRINTER SPACING CHART

IBM 407, 408, 409, 1403, 1404, 1443, and 2201

FIELD HEADINGS/WORD MARKS 6 Lines Per Inch Print span: IBM 1403 Models 1 & 4 IBM 1443 Models 1 and N1

```
 1
 2  DATE XX/XX/XX                         SALES ANALYSIS REPORT                   PAGE XXØX
 3
 4           DEPARTMENT     SALESMAN                                 CURRENT
 5               NUMBER      NUMBER            NAME                   SALES
 6
 7               XX          XXX       XXXXXXXXXXXXXXXXXXXX          XX,XXØ.XX
 8               XX          XXX       XXXXXXXXXXXXXXXXXXXX          XX,XXØ.XX
 9               XX          XXX       XXXXXXXXXXXXXXXXXXXX          XX,XXØ.XX
10
11                                 TOTAL SALES  DEPARTMENT NO XX  $XXX,XXØ.XX*
12
13
14                                                                  FINAL TOTAL $X,XXX,XXØ.XX**
15
16
```

```
DATE 10/10/72                         SALES ANALYSIS REPORT                    PAGE    7

        DEPARTMENT     SALESMAN                                 CURRENT
          NUMBER        NUMBER            NAME                   SALES

            04           120       ALBERTS, J.                   527.90
            04           121       BANKOR, F.                  2,328.50
            04           123       JERSIT, P.                 79,246.00

                      TOTAL SALES   DEPARTMENT NO 04       $ 82,102.40*


                                                                    FINAL TOTAL   $2,375,498.50**
```

TEST DATA – ASSIGNMENT NO. 1

Department (Col 3-4)	Salesman (Col 5-7)	Name (Col 8-27)	Current Sales (Col 35-41)
04	120	Alberts, J.	0052790
04	121	Bankor, F.	0232850
04	123	Jersit, P.	7924600
05	096	Peters, P.	0045692
05	236	Metyrs, T.	0987400
19	774	Snider, T.	0093101

CHAPTER 9

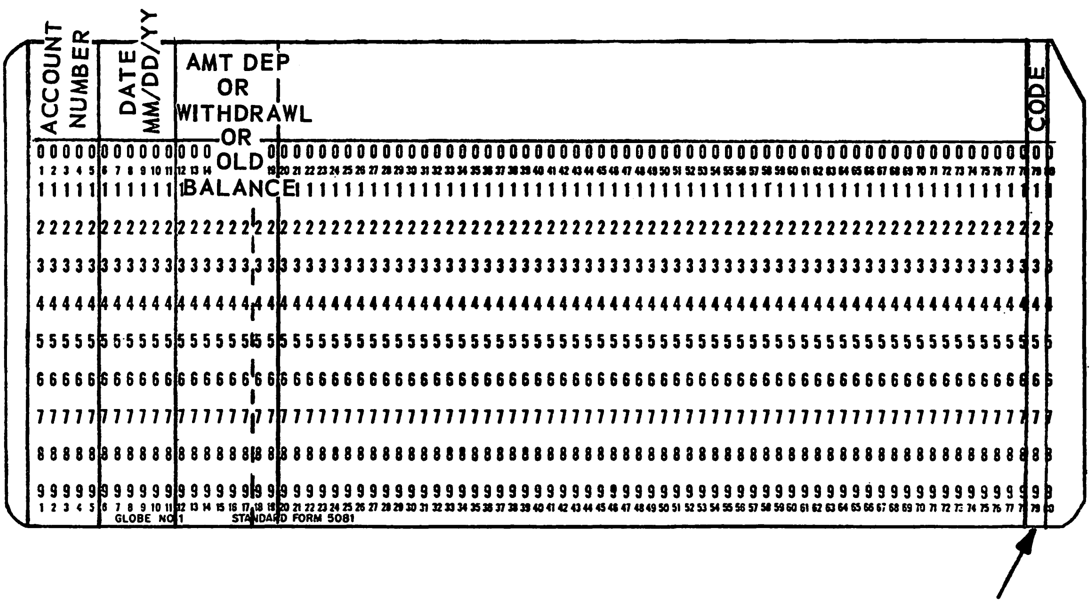
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

A group indicated Account Balance Report is to be created. From the input, a balance for each account is to be accumulated and printed.

INPUT

The input consists of a file of Transaction cards which indicate the deposits and withdrawls which have taken place for a given account. The format of the input card is illustrated below.



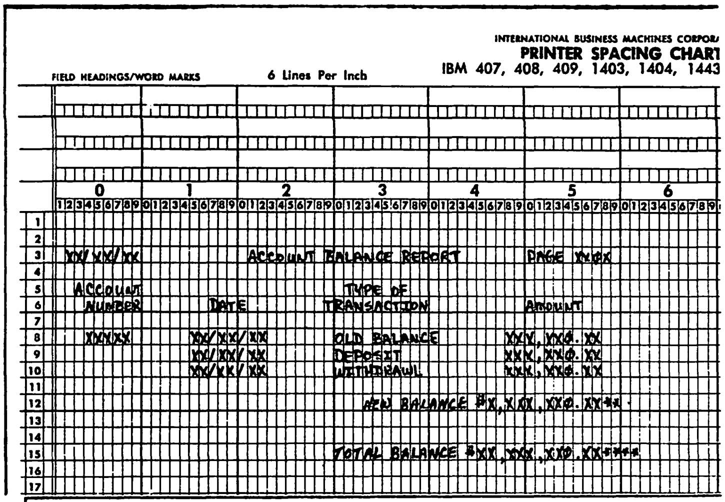
1 = OLD BALANCE
2 = WITHDRAWL
3 = DEPOSIT

In the example above it can be seen that the input card contains an Account Number, the date of the transaction, and an amount field which contains either the old balance, the amount withdrawn from the account, or the amount deposited to the account. The Code field in column 79 identifies the type of transaction – if the card contains the old balance, then the code is equal to "1", a withdrawl is identified by the code "2", and a deposit is identified by the code "3".

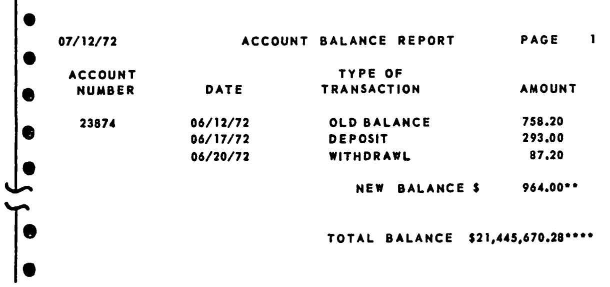
The cards are sorted in an ascending sequence by account number.

OUTPUT

The output is to consist of the Account Balance Report and a Summary card for the new balance. The format of the report is illustrated below.


INTERNATIONAL BUSINESS MACHINES CORPOR
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
0
1
2
3
4
5
6

```
3   XX/XX/XX          ACCOUNT BALANCE REPORT      PAGE XX0X

5    ACCOUNT                    TYPE OF
6     NUMBER       DATE       TRANSACTION           AMOUNT

8     XXXXX       XX/XX/XX    OLD BALANCE         XXX,XX0.XX
9                 XX/XX/XX    DEPOSIT             XXX,XX0.XX
10                XX/XX/XX    WITHDRAWL           XXX,XX0.XX

12                               NEW BALANCE $X,XXX,XX0.XX**

15                           TOTAL BALANCE $XX,XXX,XX0.XX****
```



```
07/12/72                ACCOUNT BALANCE REPORT        PAGE   1

 ACCOUNT                       TYPE OF
  NUMBER        DATE         TRANSACTION              AMOUNT

  23874       06/12/72       OLD BALANCE              758.20
              06/17/72       DEPOSIT                  293.00
              06/20/72       WITHDRAWL                 87.20

                                NEW  BALANCE $        964.00**


                             TOTAL  BALANCE  $21,445,670.28****
```

The New Balance is calculated by the following formula: New Balance = Old Balance + Deposits - Withdrawls. The Total Balance is the sum of all of the New Balances. Note that the report is group indicated by Account Number.

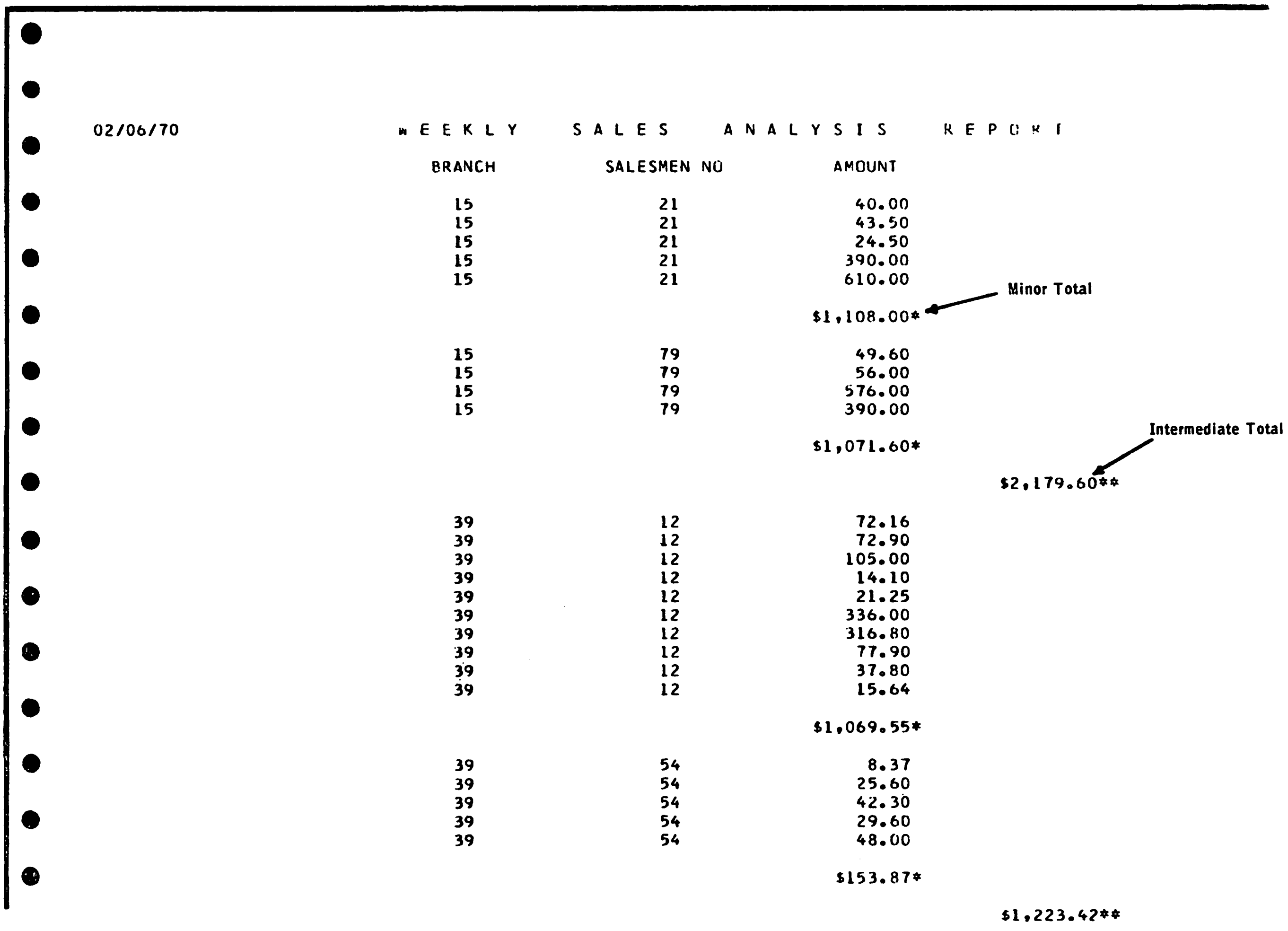
TEST DATA - ASSIGNMENT NO. 2

Account Number (col 1-5)	Date (col 6-11)	Amount (col 12-19)	Code (col 79)
23874	061272	00075820	1
23874	061772	00029300	3
23874	062072	00008720	2
45397	061272	09976356	1
45397	070172	00876567	2
67549	061272	34378398	1

CHAPTER 10

MAJOR, INTERMEDIATE, MINOR TOTALS

Once an understanding has been gained as to the basic logic necessary to take minor and final totals, the student of programming should have little difficulty in expanding this logic to take higher levels of totals. The following report illustrates an application in which a Minor Total is taken by Salesman and Intermediate Totals are taken by Branch.



```
02/06/70                  W E E K L Y      S A L E S      A N A L Y S I S      R E P O R T

                            BRANCH          SALESMEN NO          AMOUNT

                              15                21                  40.00
                              15                21                  43.50
                              15                21                  24.50
                              15                21                 390.00
                              15                21                 610.00

                                                               $1,108.00*

                              15                79                  49.60
                              15                79                  56.00
                              15                79                 576.00
                              15                79                 390.00

                                                               $1,071.60*

                                                                                   $2,179.60**

                              39                12                  72.16
                              39                12                  72.90
                              39                12                 105.00
                              39                12                  14.10
                              39                12                  21.25
                              39                12                 336.00
                              39                12                 316.80
                              39                12                  77.90
                              39                12                  37.80
                              39                12                  15.64

                                                               $1,069.55*

                              39                54                   8.37
                              39                54                  25.60
                              39                54                  42.30
                              39                54                  29.60
                              39                54                  48.00

                                                                 $153.87*

                                                                                   $1,223.42**
```

Minor Total
Intermediate Total

Figure 10-1 Sample Report with Intermediate and Minor Totals

The diagrams on the following pages illustrate the logic for taking Intermediate, Minor, and Final Totals.

Flowchart Logic - Processing The First Card

In the flowchart in Figure 10-2 the following sequence of events occurs when processing the first card.

1. A card is read, and the internal logic check for the last card is performed.
2. The Branch Number from the first card is compared to a "Save-Area". As the Save-Area will contain zeros when the first card is read and compared, a branch occurs to the SWITCH1-ROUTINE.
3. As the Switch is OFF on the first card, a branch occurs to the routine to turn the Switch ON.
4. The Branch Number and Salesman Number are moved to the Save-Area.
5. A branch occurs to the ADD-ROUTINE, where the fields to be printed are moved to the print area and the amount from the first card is added to the Minor Total Counter.
6. A line is printed and a branch occurs to read the next card. If the next card contains a Branch Number and Salesman Number equal to the first card, the Amount from that card is added to the Minor Total Counter and a line is printed. This sequence of events continues as long as the Branch Number and Salesman Number on subsequent cards are equal to to those of the first card.

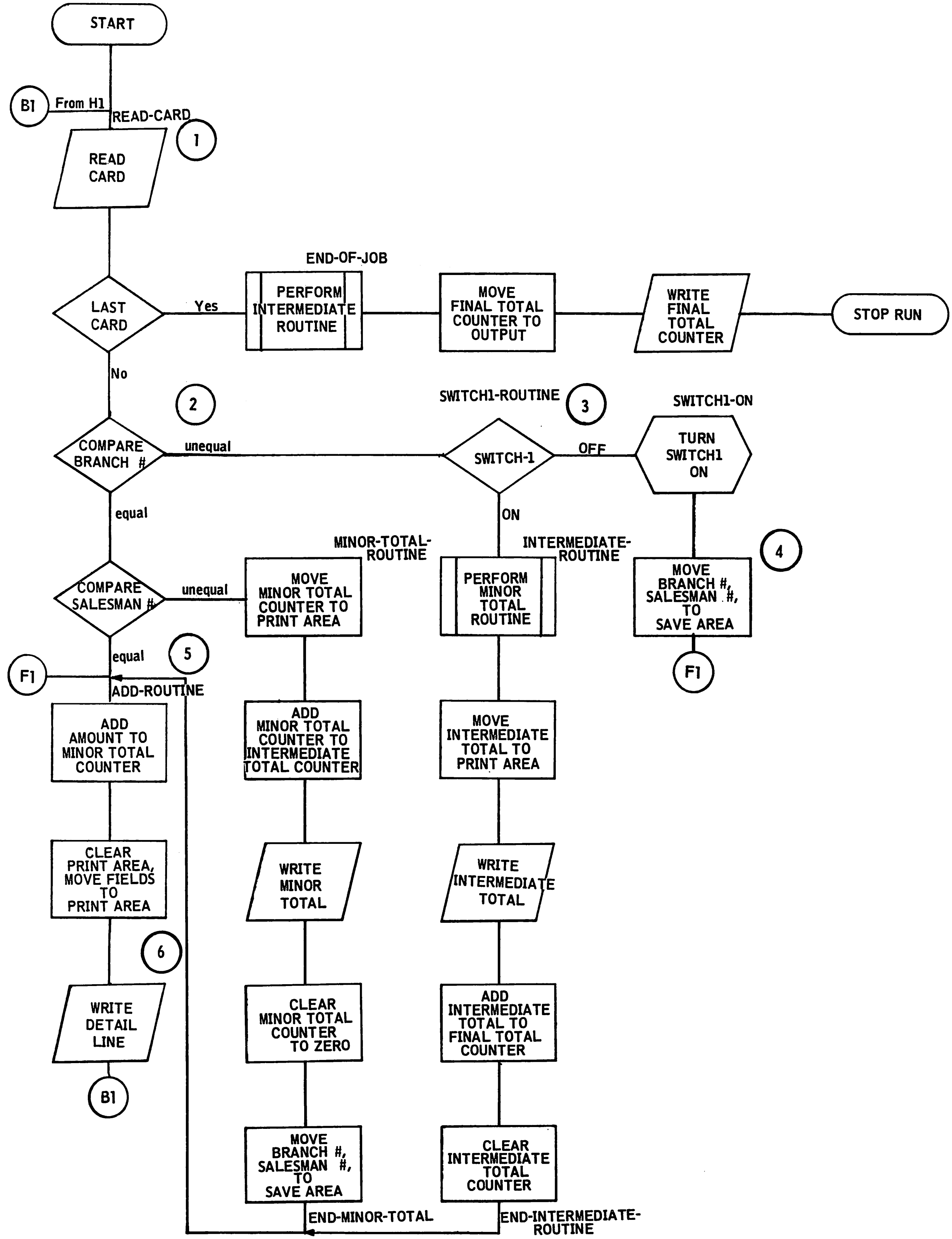

START
B1
From H1
READ-CARD
1
READ CARD
LAST CARD
Yes
END-OF-JOB
PERFORM INTERMEDIATE ROUTINE
MOVE FINAL TOTAL COUNTER TO OUTPUT
WRITE FINAL TOTAL COUNTER
STOP RUN
No
2
COMPARE BRANCH #
unequal
SWITCH1-ROUTINE
3
SWITCH-1
OFF
SWITCH1-ON
TURN SWITCH1 ON
equal
ON
4
MINOR-TOTAL-ROUTINE
INTERMEDIATE-ROUTINE
COMPARE SALESMAN #
unequal
MOVE MINOR TOTAL COUNTER TO PRINT AREA
PERFORM MINOR TOTAL ROUTINE
MOVE BRANCH #, SALESMAN #, TO SAVE AREA
equal
5
F1
ADD-ROUTINE
F1
ADD AMOUNT TO MINOR TOTAL COUNTER
ADD MINOR TOTAL COUNTER TO INTERMEDIATE TOTAL COUNTER
MOVE INTERMEDIATE TOTAL TO PRINT AREA
CLEAR PRINT AREA, MOVE FIELDS TO PRINT AREA
WRITE MINOR TOTAL
WRITE INTERMEDIATE TOTAL
6
WRITE DETAIL LINE
CLEAR MINOR TOTAL COUNTER TO ZERO
ADD INTERMEDIATE TOTAL TO FINAL TOTAL COUNTER
B1
MOVE BRANCH #, SALESMAN #, TO SAVE AREA
CLEAR INTERMEDIATE TOTAL COUNTER
END-MINOR-TOTAL
END-INTERMEDIATE-ROUTINE

Figure 10-2 Flowchart – Intermediate and Minor Total Logic for processing of the first card

Flowchart Logic - Minor Control Change

The Flowchart in Figure 10-3 illustrates the basic sequence of events occurring when a card is read that contains a Salesman Number that is not equal to the first card of the control group.

1. Cards are compared and the Salesman Number is unequal.
2. The Minor Total Counter is moved to the print area.
3. Contents of the Minor Total Counter are added to the Intermediate Counter.
4. Minor Total is printed.
5. Minor Total Counter is cleared to zero.
6. The Branch Number and Salesman Number are moved to the Save-Area.
7. A branch occurs to the ADD-ROUTINE where the fields to be printed are moved to the print area and the amount for the new Salesman is added to the Minor Total Counter.
8. A line is printed.

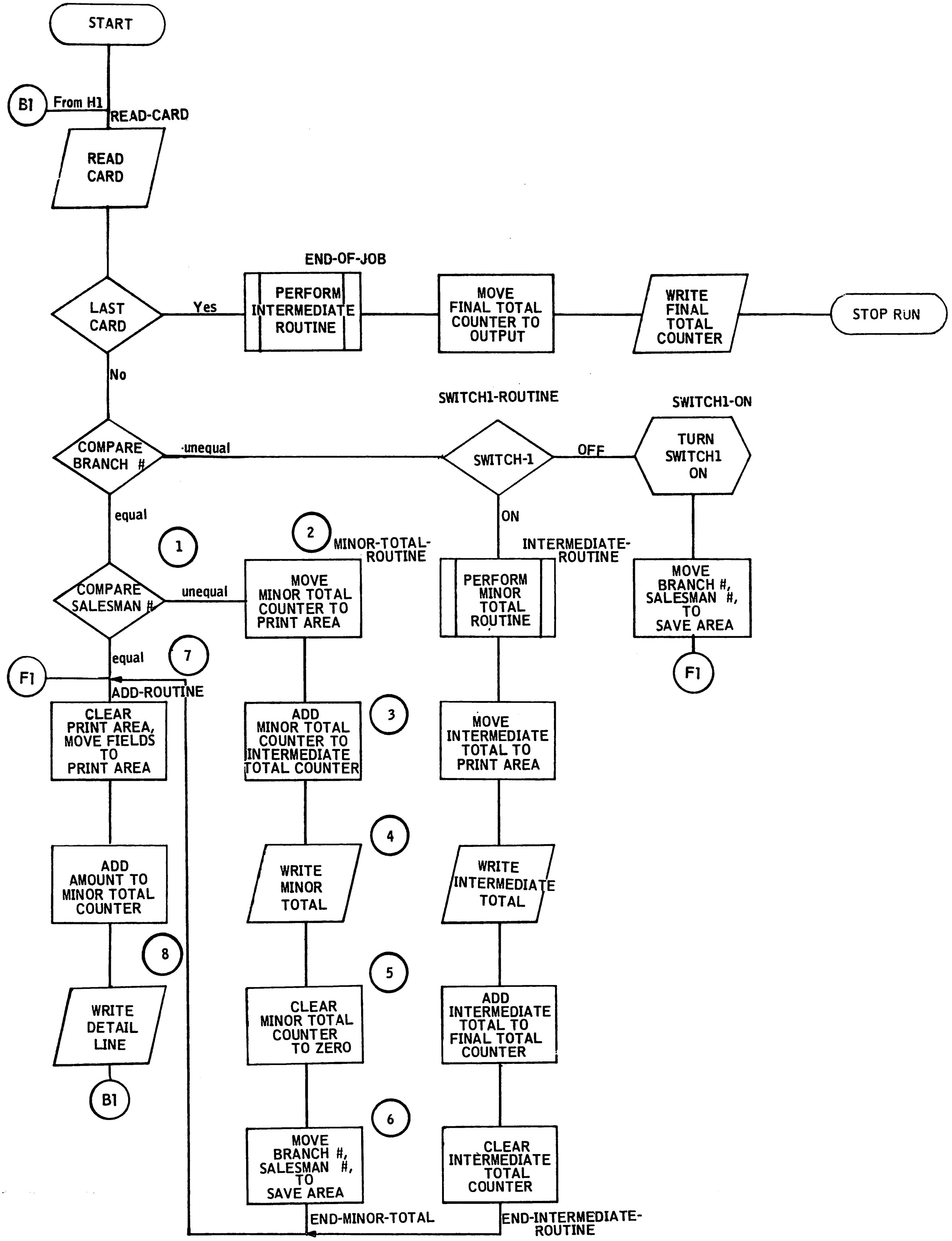

START
B1
From H1
READ-CARD
READ CARD
LAST CARD
Yes
END-OF-JOB
PERFORM INTERMEDIATE ROUTINE
MOVE FINAL TOTAL COUNTER TO OUTPUT
WRITE FINAL TOTAL COUNTER
STOP RUN
No
COMPARE BRANCH #
unequal
SWITCH1-ROUTINE
SWITCH-1
OFF
SWITCH1-ON
TURN SWITCH1 ON
equal
ON
MOVE BRANCH #, SALESMAN #, TO SAVE AREA
F1
1
2
MINOR-TOTAL-ROUTINE
INTERMEDIATE-ROUTINE
COMPARE SALESMAN #
unequal
MOVE MINOR TOTAL COUNTER TO PRINT AREA
PERFORM MINOR TOTAL ROUTINE
equal
7
F1
ADD-ROUTINE
CLEAR PRINT AREA, MOVE FIELDS TO PRINT AREA
ADD MINOR TOTAL COUNTER TO INTERMEDIATE TOTAL COUNTER
3
MOVE INTERMEDIATE TOTAL TO PRINT AREA
4
ADD AMOUNT TO MINOR TOTAL COUNTER
WRITE MINOR TOTAL
WRITE INTERMEDIATE TOTAL
8
5
WRITE DETAIL LINE
CLEAR MINOR TOTAL COUNTER TO ZERO
ADD INTERMEDIATE TOTAL TO FINAL TOTAL COUNTER
B1
6
MOVE BRANCH #, SALESMAN #, TO SAVE AREA
CLEAR INTERMEDIATE TOTAL COUNTER
END-MINOR-TOTAL
END-INTERMEDIATE-ROUTINE

Figure 10-3 Flowchart – Minor Control Change

Flowchart Logic - Intermediate Control Change

The Flowchart in Figure 10-4 illustrates the basic sequence of events occuring when a card is read in which the Branch Number is not equal to the Branch Number of the previous card group.

1. Cards are compared and the Branch Numbers are unequal.
2. A branch occurs to the SWITCH1-ROUTINE. As the Switch is ON, the INTERMEDIATE-ROUTINE is executed.
3. When an Intermediate Total is taken, a Minor Total must also be taken. Thus, a statement is written in the COBOL Program to Perform the MINOR-TOTAL-ROUTINE. This routine prints the Minor Total from the previous card group, adds the contents of the Minor Total Counter to the Intermediate Counter, clears the Minor Total Counter, Moves the Branch Number and Salesman Number to the Save-Area, then returns to the instruction following the Perform.
4. The contents of the Intermediate Counter is moved to the Print Area and printed.
5. The contents of the Intermediate Counter is added to the Final Total Counter. Thus, the Final Total will be a sum of the contents of the Intermediate Counter.
6. The Intermediate Counter is cleared to zero, and a branch occurs to the ADD-ROUTINE.
7. The fields to be printed are moved to the print area and the Amount Field from the card that caused the "control break" is added to the Minor Total Counter.
8. A line is printed, and the sequence of operations is repeated.

When the last card is read, a branch occurs to the END-OF-JOB ROUTINE. At the END-OF-JOB ROUTINE, a Statement is given in the COBOL program to perform the INTERMEDIATE-ROUTINE. When the INTERMEDIATE-ROUTINE is executed, the MINOR-ROUTINE will also be performed. Thus, both the Minor Totals and Intermediate Totals will be printed. When the INTERMEDIATE-ROUTINE has been performed, the program will return to the instruction following the Perform, which moves the Final Total to the Output Area and prints the Final Total. The execution of the program is then terminated.

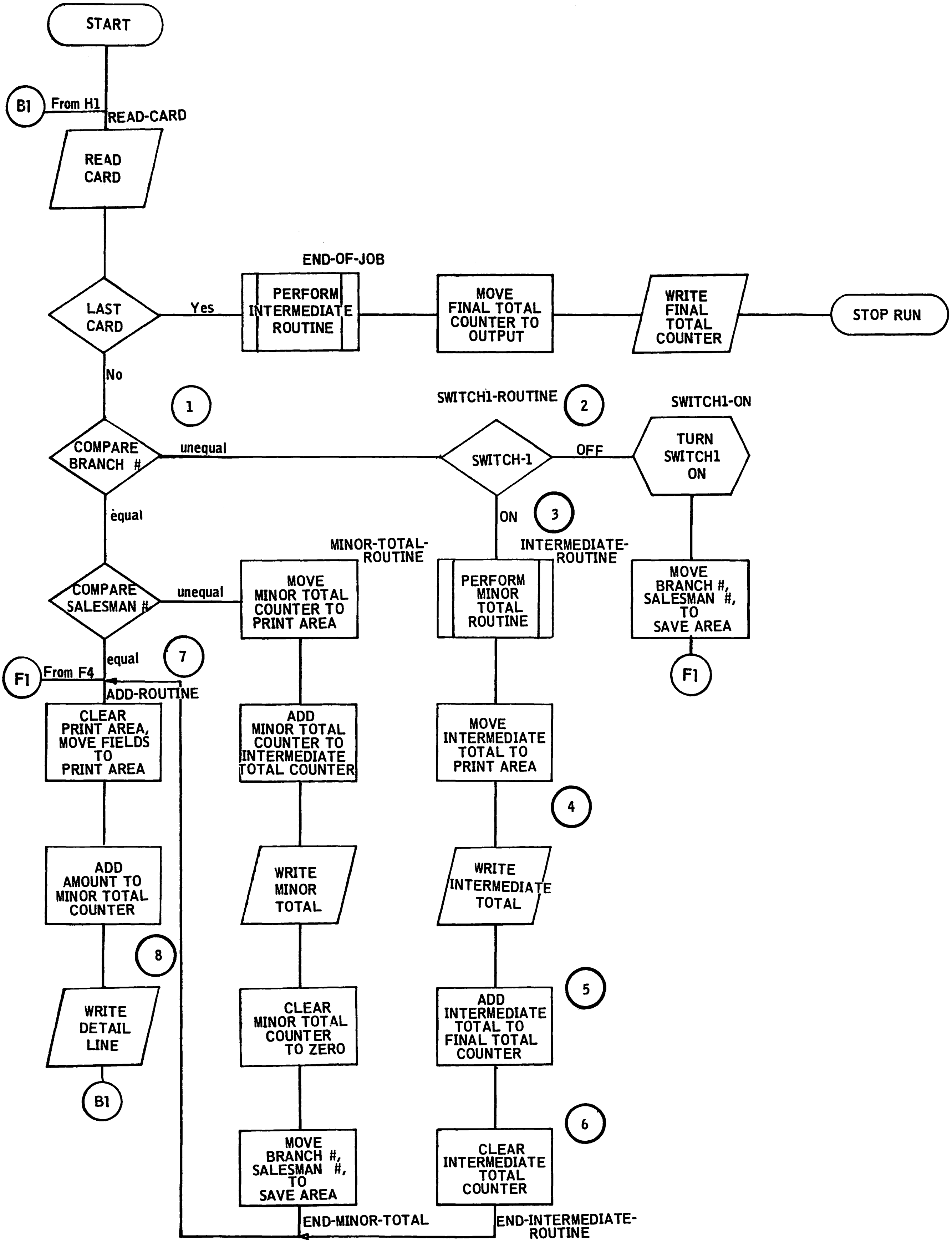

START
B1
From H1
READ-CARD
READ CARD
LAST CARD
Yes
END-OF-JOB
PERFORM INTERMEDIATE ROUTINE
MOVE FINAL TOTAL COUNTER TO OUTPUT
WRITE FINAL TOTAL COUNTER
STOP RUN
No
1
COMPARE BRANCH #
unequal
SWITCH1-ROUTINE
2
SWITCH-1
OFF
SWITCH1-ON
TURN SWITCH1 ON
equal
ON
3
MINOR-TOTAL-ROUTINE
INTERMEDIATE-ROUTINE
COMPARE SALESMAN #
unequal
MOVE MINOR TOTAL COUNTER TO PRINT AREA
PERFORM MINOR TOTAL ROUTINE
MOVE BRANCH #, SALESMAN #, TO SAVE AREA
F1
equal
7
F1
From F4
ADD-ROUTINE
CLEAR PRINT AREA, MOVE FIELDS TO PRINT AREA
ADD MINOR TOTAL COUNTER TO INTERMEDIATE TOTAL COUNTER
MOVE INTERMEDIATE TOTAL TO PRINT AREA
4
ADD AMOUNT TO MINOR TOTAL COUNTER
WRITE MINOR TOTAL
WRITE INTERMEDIATE TOTAL
8
5
WRITE DETAIL LINE
CLEAR MINOR TOTAL COUNTER TO ZERO
ADD INTERMEDIATE TOTAL TO FINAL TOTAL COUNTER
B1
6
MOVE BRANCH #, SALESMAN #, TO SAVE AREA
CLEAR INTERMEDIATE TOTAL COUNTER
END-MINOR-TOTAL
END-INTERMEDIATE-ROUTINE

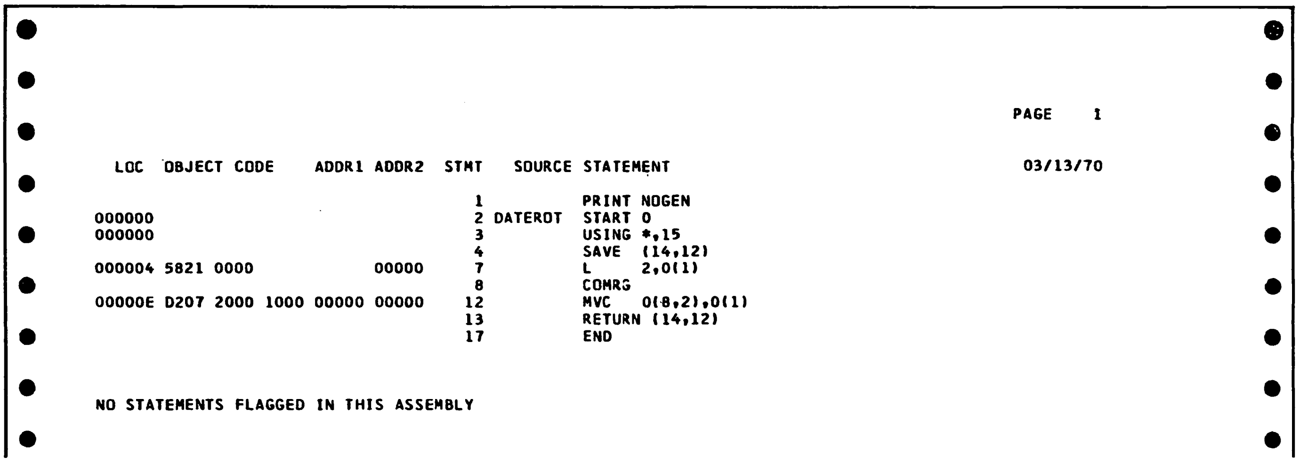
Figure 10-4 Flowchart – Intermediate Control Change

SUB-PROGRAMS

In some applications, it may be desirable to write certain segments of the program in a language other than COBOL. The COBOL language provides the capability of "exiting" from the COBOL program to a routine written in another programming language and then returning to the main COBOL program. This capability provides flexibility for performing certain types of processing which are awkward or cannot be done in COBOL using some other programming language.

One application of a sub-program written in another programming language is illustrated in the sample program at the end of the chapter. It uses a subroutine written in Assembler Language to extract the date from the computer system. Although this subroutine is written for the System/360, the mechanism for calling a subroutine written in another programming language is common for all programs written in ANSI COBOL.

The subroutine which is called is an Assembler Language subroutine and is illustrated below merely for purposes of showing the type of subroutine which may be used.



```
                                                                                  PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                       03/13/70

                                     1          PRINT NOGEN
000000                               2 DATEROT  START 0
000000                               3          USING *,15
                                     4          SAVE  (14,12)
000004 5821 0000            00000    7          L     2,0(1)
                                     8          COMRG
00000E D207 2000 1000 00000 00000   12          MVC   0(8,2),0(1)
                                    13          RETURN (14,12)
                                    17          END


NO STATEMENTS FLAGGED IN THIS ASSEMBLY
```

Figure 10-5 Assembler Sub-Program

The purpose of this routine is to make the current date available for printing in the title of the report.

A COBOL source program which passes control to another program is referred to as a "calling program". The program which receives control from the calling program is referred to as a "called program". Both programs must be compiled tn separate job steps, but the resulting object modules must be "linked" together into a single program.

Whenever a program calls another program, a link must be established between the two. The calling program must state the entry point of the called program, and must specify any addresses (called arguments) to be passed. The "called program" must have an entry point and must be able to accept the addresses (arguments). Further, the called program must establish the linkage for the return of control to the calling program.

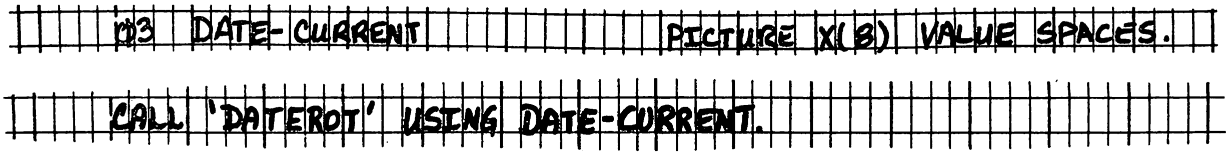
A calling COBOL program must contain the following statements at the point where another program is to be called.

```
CALL literal [USING identifier-1 [identifier-2] ...]
```

Figure 10-6 Format Notation

Note in the example in Figure 10-6 that the verb CALL is specified followed by a "literal". The literal is a non-numeric literal which, therefore, must be enclosed within apostrophes and is the name of the program to be called or the entry point within a program to which control is to be passed. The USING clause is used to specify identifiers or data-names defined within the COBOL program which are to be referenced within the sub-program. One or more identifiers may be specified.

The segment of the COBOL program used to exit from the COBOL program and enter the Assembler Language routine and return to the main COBOL routine is illustrated below.



```
    03  DATE-CURRENT               PICTURE X(8) VALUE SPACES.
    CALL 'DATEROT' USING DATE-CURRENT.
```

Figure 10-7 COBOL Calling Program

In the example in Figure 10-7 it can be seen that the non-numeric literal specified is DATEROT, which is the name of the entry point in the subprogram. The field DATE-CURRENT is defined within the Title Line for the report and is to contain the current date for printing on the report. Thus, when the DATEROT routine has completed its processing, it is desired that the current date be placed in the DATE-CURRENT field. Therefore, the address of the field is passed to the subprogram through the use of the Using statement as illustrated.

Thus, the effect of the calling of the sub-program is to have the sub-program extract the date and place it in the field DATE-CURRENT which is defined within the COBOL program.

NOTE Statement

The NOTE Statement permits the programmer to write explanatory comments, in the Procedure Division of a source program, which will be produced on the listing but serve no other purpose. The format of the NOTE Statement is:

```
NOTE comment...
```

Figure 10-8 Note Statement

NOTE, when used, must begin a sentence. Following the word NOTE, any combination of the characters from the COBOL character set may appear. If NOTE is the first word of a paragraph, any remaining sentences within the paragraph are also considered notes. Proper format rules for paragraph structure must be observed.

SAMPLE PROBLEM

A further extension of the logic presented in the previous examples involves an application in which three levels of totals are taken: Major, Intermediate, and Minor. In addition, the PERFORM statement is quite often used to cause the total routines to be processed instead of GO TO statements.

The following sample problem illustrates the input, output, flowchart, and COBOL program for the taking of Major, Intermediate, and Minor Totals. A Final Total is also taken after all cards have been processed. A Major Total is to be taken by Date, an Intermediate Total by Branch, and a Minor Total by Salesman. The Perform statement is used in the sample program to process the total routines.

Input

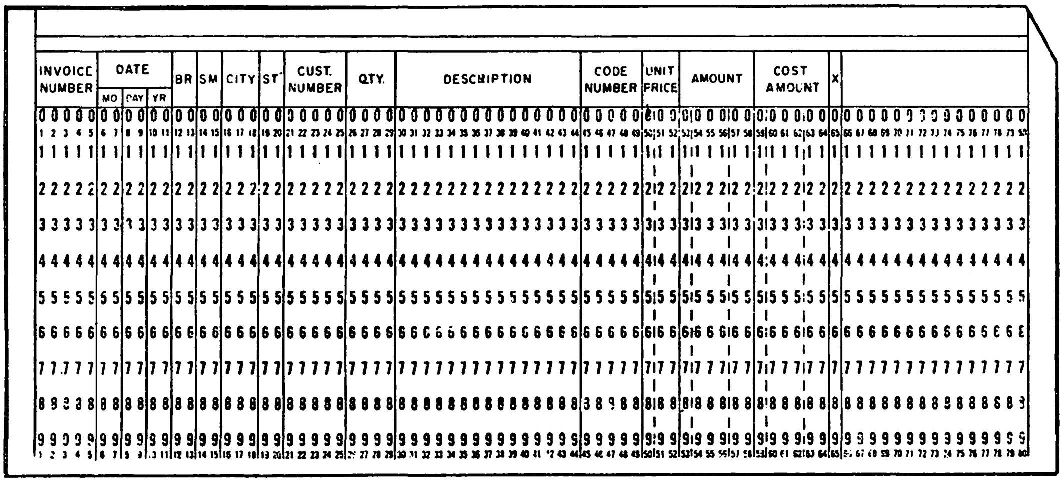
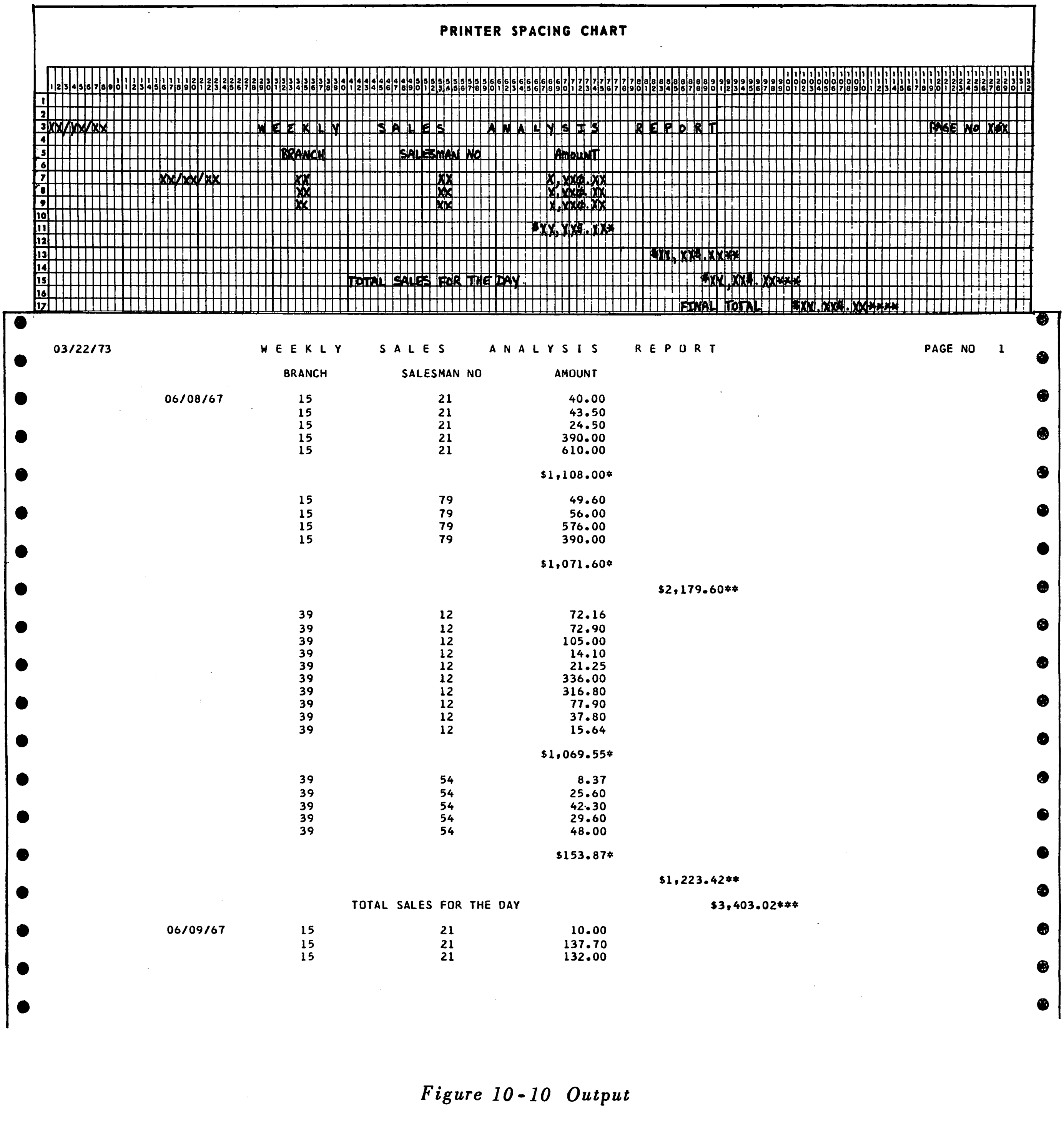


Figure 10-9 Input

Output

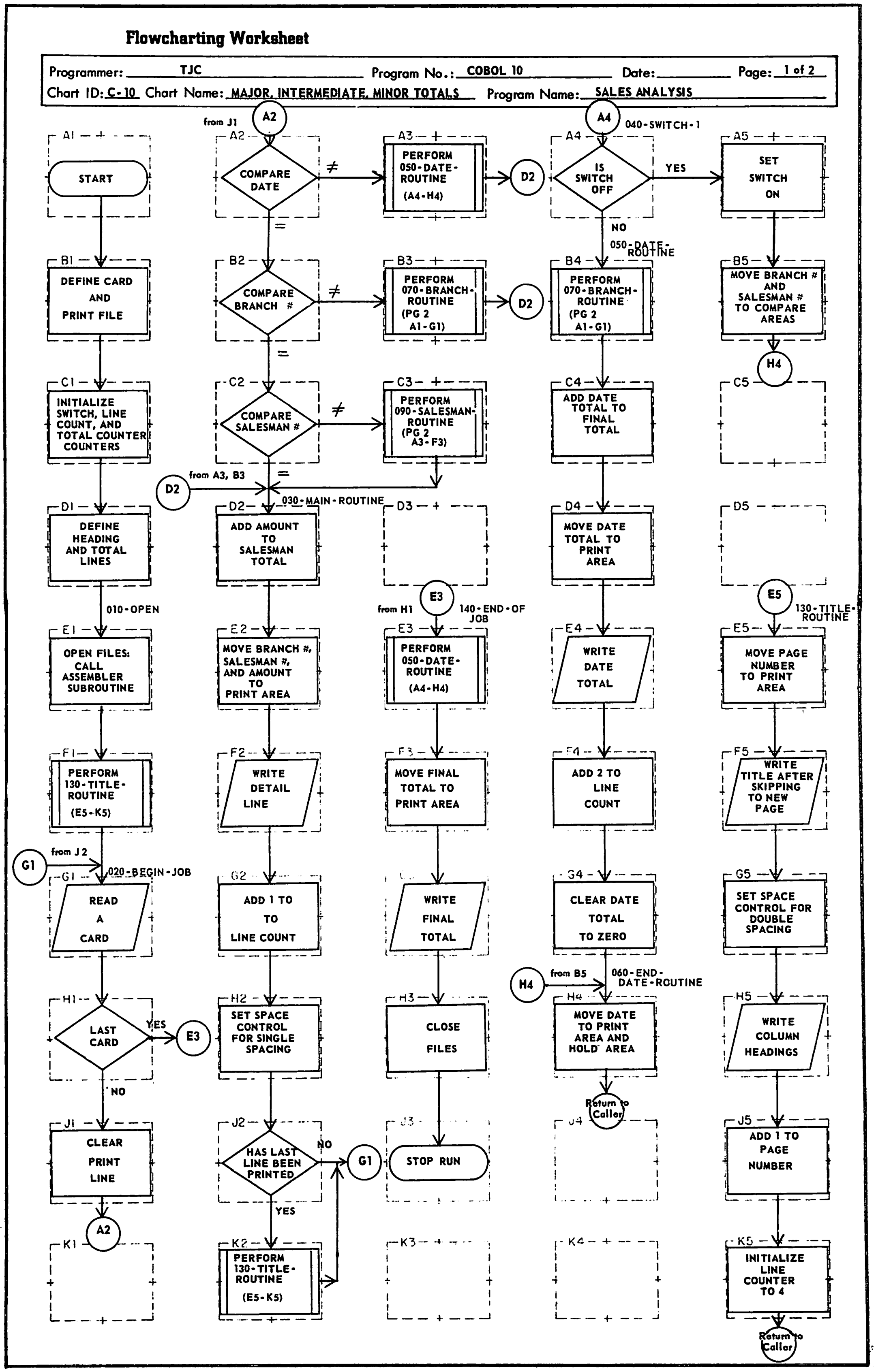


```
PRINTER SPACING CHART

 3 XX/XX/XX               W E E K L Y   S A L E S   A N A L Y S I S   R E P O R T                PAGE NO X0X
 5                          BRANCH          SALESMAN NO        AMOUNT
 7            XX/XX/XX        XX                XX           X,XX0.XX
 8                            XX                XX           X,XX0.XX
 9                            XX                XX           X,XX0.XX
11                                                         $XX,XX0.XX*
13                                                                        $XX,XX0.XX**
15                                TOTAL SALES FOR THE DAY                      $XX,XX0.XX***
17                                                                     FINAL TOTAL    $XX,XX0.XX****
```

```
03/22/73                   W E E K L Y     S A L E S     A N A L Y S I S     R E P O R T              PAGE NO   1

                            BRANCH          SALESMAN NO          AMOUNT

          06/08/67            15                21                40.00
                              15                21                43.50
                              15                21                24.50
                              15                21               390.00
                              15                21               610.00

                                                              $1,108.00*

                              15                79                49.60
                              15                79                56.00
                              15                79               576.00
                              15                79               390.00

                                                              $1,071.60*

                                                                           $2,179.60**

                              39                12                72.16
                              39                12                72.90
                              39                12               105.00
                              39                12                14.10
                              39                12                21.25
                              39                12               336.00
                              39                12               316.80
                              39                12                77.90
                              39                12                37.80
                              39                12                15.64

                                                              $1,069.55*

                              39                54                 8.37
                              39                54                25.60
                              39                54                42.30
                              39                54                29.60
                              39                54                48.00

                                                                $153.87*

                                                                           $1,223.42**

                                      TOTAL SALES FOR THE DAY                      $3,403.02***

          06/09/67            15                21                10.00
                              15                21               137.70
                              15                21               132.00
```

Figure 10-10 Output

Flowchart



Flowcharting Worksheet

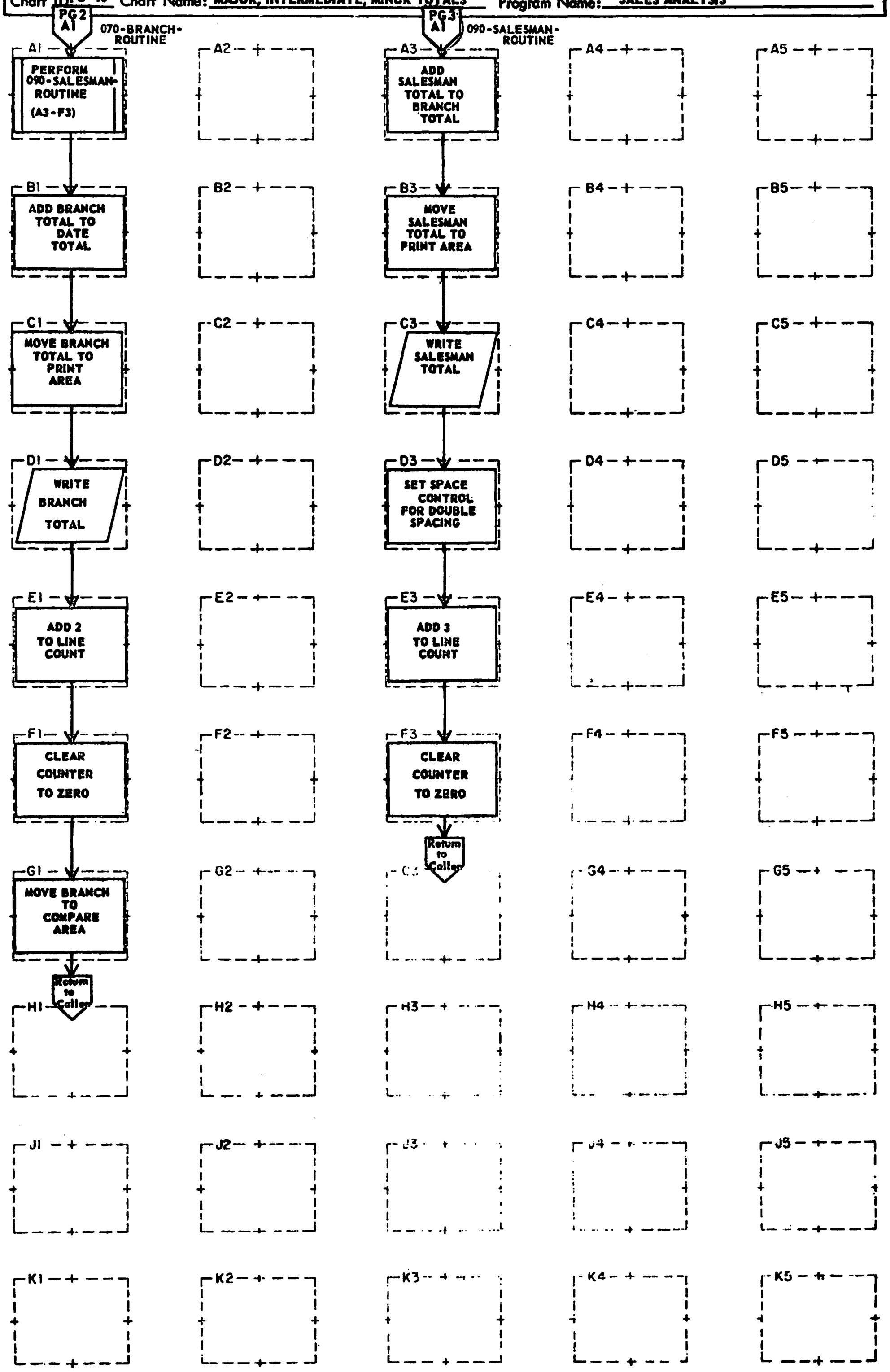
Programmer: TJC Program No.: COBOL 10 Date: Page: 1 of 2
Chart ID: C-10 Chart Name: MAJOR, INTERMEDIATE, MINOR TOTALS Program Name: SALES ANALYSIS

A1 START
B1 DEFINE CARD AND PRINT FILE
C1 INITIALIZE SWITCH, LINE COUNT, AND TOTAL COUNTER COUNTERS
D1 DEFINE HEADING AND TOTAL LINES
010-OPEN
E1 OPEN FILES: CALL ASSEMBLER SUBROUTINE
F1 PERFORM 130-TITLE-ROUTINE (E5-K5)
G1 from J2
020-BEGIN-JOB
G1 READ A CARD
H1 LAST CARD
YES
E3
NO
J1 CLEAR PRINT LINE
A2
K1
from J1
A2
A2 COMPARE DATE
≠
=
B2 COMPARE BRANCH #
≠
=
C2 COMPARE SALESMAN #
≠
=
D2 from A3, B3
030-MAIN-ROUTINE
D2 ADD AMOUNT TO SALESMAN TOTAL
E2 MOVE BRANCH #, SALESMAN #, AND AMOUNT TO PRINT AREA
F2 WRITE DETAIL LINE
G2 ADD 1 TO LINE COUNT
H2 SET SPACE CONTROL FOR SINGLE SPACING
J2 HAS LAST LINE BEEN PRINTED
NO
G1
YES
K2 PERFORM 130-TITLE-ROUTINE (E5-K5)
A3 PERFORM 050-DATE-ROUTINE (A4-H4)
D2
B3 PERFORM 070-BRANCH-ROUTINE (PG 2 A1-G1)
D2
C3 PERFORM 090-SALESMAN-ROUTINE (PG 2 A3-F3)
D3
E3 from H1
140-END-OF JOB
E3 PERFORM 050-DATE-ROUTINE (A4-H4)
F3 MOVE FINAL TOTAL TO PRINT AREA
G3 WRITE FINAL TOTAL
H3 CLOSE FILES
J3 STOP RUN
K3
A4 040-SWITCH-1
A4 IS SWITCH OFF
YES
NO
050-DATE-ROUTINE
B4 PERFORM 070-BRANCH-ROUTINE (PG 2 A1-G1)
C4 ADD DATE TOTAL TO FINAL TOTAL
D4 MOVE DATE TOTAL TO PRINT AREA
E4 WRITE DATE TOTAL
F4 ADD 2 TO LINE COUNT
G4 CLEAR DATE TOTAL TO ZERO
H4 from B5
060-END-DATE-ROUTINE
H4 MOVE DATE TO PRINT AREA AND HOLD AREA
Return to Caller
J4
K4
A5 SET SWITCH ON
B5 MOVE BRANCH # AND SALESMAN # TO COMPARE AREAS
H4
C5
D5
E5 130-TITLE-ROUTINE
E5 MOVE PAGE NUMBER TO PRINT AREA
F5 WRITE TITLE AFTER SKIPPING TO NEW PAGE
G5 SET SPACE CONTROL FOR DOUBLE SPACING
H5 WRITE COLUMN HEADINGS
J5 ADD 1 TO PAGE NUMBER
K5 INITIALIZE LINE COUNTER TO 4
Return to Caller

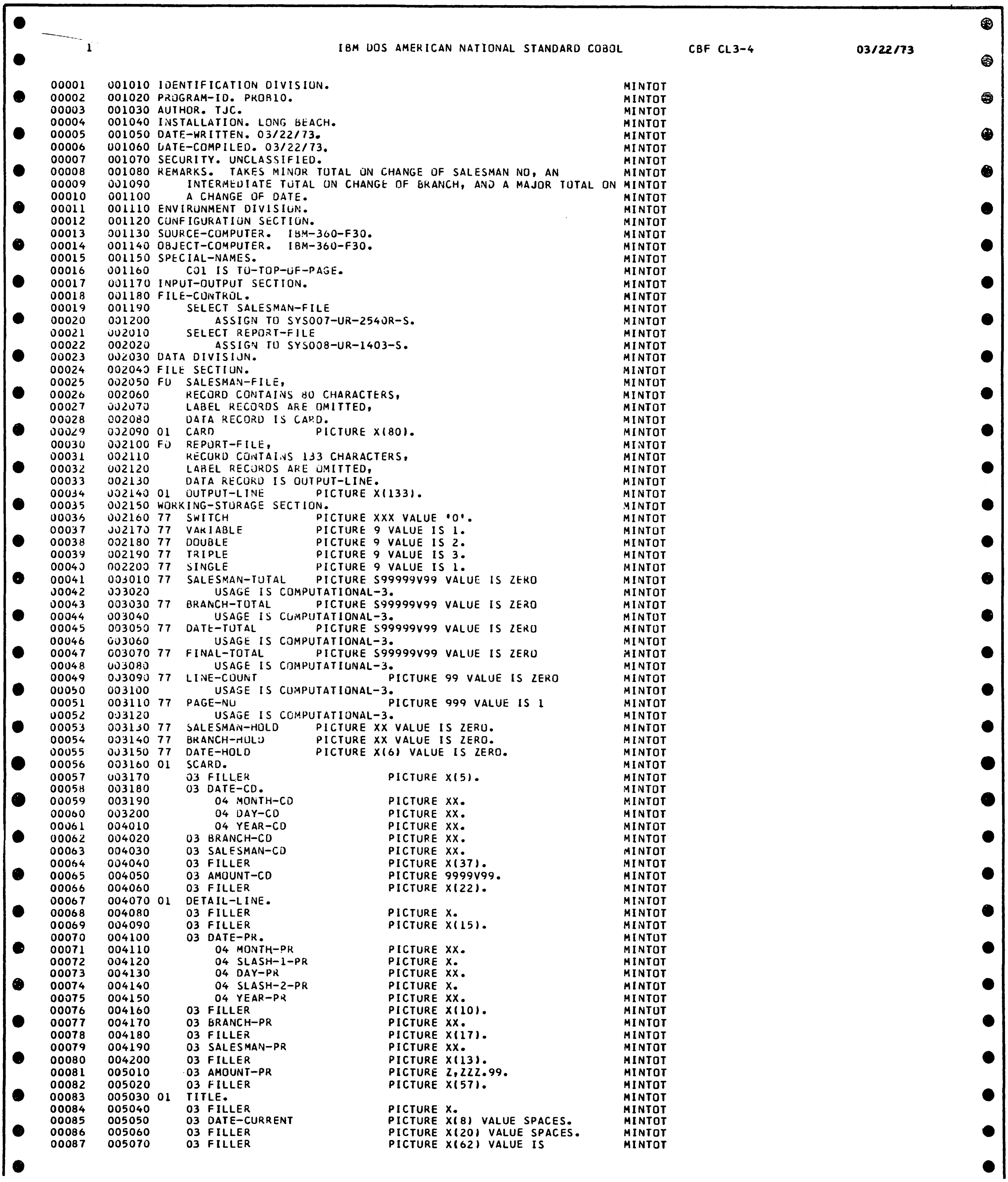
Figure 10-11 Flowchart

Flowcharting Worksheet

Programmer: TJC Program No.: COBOL 10 Date: ________ Page: 2 of 2
Chart ID: C-10 Chart Name: MAJOR, INTERMEDIATE, MINOR TOTALS Program Name: SALES ANALYSIS


PG2 A1
070-BRANCH-ROUTINE
A1
PERFORM 090-SALESMAN-ROUTINE (A3-F3)
B1
ADD BRANCH TOTAL TO DATE TOTAL
C1
MOVE BRANCH TOTAL TO PRINT AREA
D1
WRITE BRANCH TOTAL
E1
ADD 2 TO LINE COUNT
F1
CLEAR COUNTER TO ZERO
G1
MOVE BRANCH TO COMPARE AREA
Return to Caller
PG3 A1
090-SALESMAN-ROUTINE
A3
ADD SALESMAN TOTAL TO BRANCH TOTAL
B3
MOVE SALESMAN TOTAL TO PRINT AREA
C3
WRITE SALESMAN TOTAL
D3
SET SPACE CONTROL FOR DOUBLE SPACING
E3
ADD 3 TO LINE COUNT
F3
CLEAR COUNTER TO ZERO
Return to Caller

SAMPLE PROGRAM



```
       1                                     IBM DOS AMERICAN NATIONAL STANDARD COBOL         CBF CL3-4           03/22/73

00001   001010 IDENTIFICATION DIVISION.                                         MINTOT
00002   001020 PROGRAM-ID. PROB10.                                              MINTOT
00003   001030 AUTHOR. TJC.                                                     MINTOT
00004   001040 INSTALLATION. LONG BEACH.                                        MINTOT
00005   001050 DATE-WRITTEN. 03/22/73.                                          MINTOT
00006   001060 DATE-COMPILED. 03/22/73.                                         MINTOT
00007   001070 SECURITY. UNCLASSIFIED.                                          MINTOT
00008   001080 REMARKS.  TAKES MINOR TOTAL ON CHANGE OF SALESMAN NO, AN         MINTOT
00009   001090     INTERMEDIATE TOTAL ON CHANGE OF BRANCH, AND A MAJOR TOTAL ON MINTOT
00010   001100     A CHANGE OF DATE.                                            MINTOT
00011   001110 ENVIRONMENT DIVISION.                                            MINTOT
00012   001120 CONFIGURATION SECTION.                                           MINTOT
00013   001130 SOURCE-COMPUTER.  IBM-360-F30.                                   MINTOT
00014   001140 OBJECT-COMPUTER.  IBM-360-F30.                                   MINTOT
00015   001150 SPECIAL-NAMES.                                                   MINTOT
00016   001160     C01 IS TO-TOP-OF-PAGE.                                       MINTOT
00017   001170 INPUT-OUTPUT SECTION.                                            MINTOT
00018   001180 FILE-CONTROL.                                                    MINTOT
00019   001190     SELECT SALESMAN-FILE                                         MINTOT
00020   001200         ASSIGN TO SYS007-UR-2540R-S.                             MINTOT
00021   002010     SELECT REPORT-FILE                                           MINTOT
00022   002020         ASSIGN TO SYS008-UR-1403-S.                              MINTOT
00023   002030 DATA DIVISION.                                                   MINTOT
00024   002040 FILE SECTION.                                                    MINTOT
00025   002050 FD  SALESMAN-FILE,                                               MINTOT
00026   002060     RECORD CONTAINS 80 CHARACTERS,                               MINTOT
00027   002070     LABEL RECORDS ARE OMITTED,                                   MINTOT
00028   002080     DATA RECORD IS CARD.                                         MINTOT
00029   002090 01  CARD              PICTURE X(80).                             MINTOT
00030   002100 FD  REPORT-FILE,                                                 MINTOT
00031   002110     RECORD CONTAINS 133 CHARACTERS,                              MINTOT
00032   002120     LABEL RECORDS ARE OMITTED,                                   MINTOT
00033   002130     DATA RECORD IS OUTPUT-LINE.                                  MINTOT
00034   002140 01  OUTPUT-LINE       PICTURE X(133).                            MINTOT
00035   002150 WORKING-STORAGE SECTION.                                         MINTOT
00036   002160 77  SWITCH            PICTURE XXX VALUE '0'.                     MINTOT
00037   002170 77  VARIABLE          PICTURE 9 VALUE IS 1.                      MINTOT
00038   002180 77  DOUBLE            PICTURE 9 VALUE IS 2.                      MINTOT
00039   002190 77  TRIPLE            PICTURE 9 VALUE IS 3.                      MINTOT
00040   002200 77  SINGLE            PICTURE 9 VALUE IS 1.                      MINTOT
00041   003010 77  SALESMAN-TOTAL    PICTURE S99999V99 VALUE IS ZERO            MINTOT
00042   003020         USAGE IS COMPUTATIONAL-3.                                MINTOT
00043   003030 77  BRANCH-TOTAL      PICTURE S99999V99 VALUE IS ZERO            MINTOT
00044   003040         USAGE IS COMPUTATIONAL-3.                                MINTOT
00045   003050 77  DATE-TOTAL        PICTURE S99999V99 VALUE IS ZERO            MINTOT
00046   003060         USAGE IS COMPUTATIONAL-3.                                MINTOT
00047   003070 77  FINAL-TOTAL       PICTURE S99999V99 VALUE IS ZERO            MINTOT
00048   003080         USAGE IS COMPUTATIONAL-3.                                MINTOT
00049   003090 77  LINE-COUNT                  PICTURE 99 VALUE IS ZERO         MINTOT
00050   003100         USAGE IS COMPUTATIONAL-3.                                MINTOT
00051   003110 77  PAGE-NO                     PICTURE 999 VALUE IS 1           MINTOT
00052   003120         USAGE IS COMPUTATIONAL-3.                                MINTOT
00053   003130 77  SALESMAN-HOLD     PICTURE XX VALUE IS ZERO.                  MINTOT
00054   003140 77  BRANCH-HOLD       PICTURE XX VALUE IS ZERO.                  MINTOT
00055   003150 77  DATE-HOLD         PICTURE X(6) VALUE IS ZERO.                MINTOT
00056   003160 01  SCARD.                                                       MINTOT
00057   003170     03 FILLER                   PICTURE X(5).                    MINTOT
00058   003180     03 DATE-CD.                                                  MINTOT
00059   003190         04 MONTH-CD             PICTURE XX.                      MINTOT
00060   003200         04 DAY-CD               PICTURE XX.                      MINTOT
00061   004010         04 YEAR-CD              PICTURE XX.                      MINTOT
00062   004020     03 BRANCH-CD                PICTURE XX.                      MINTOT
00063   004030     03 SALESMAN-CD              PICTURE XX.                      MINTOT
00064   004040     03 FILLER                   PICTURE X(37).                   MINTOT
00065   004050     03 AMOUNT-CD                PICTURE 9999V99.                 MINTOT
00066   004060     03 FILLER                   PICTURE X(22).                   MINTOT
00067   004070 01  DETAIL-LINE.                                                 MINTOT
00068   004080     03 FILLER                   PICTURE X.                       MINTOT
00069   004090     03 FILLER                   PICTURE X(15).                   MINTOT
00070   004100     03 DATE-PR.                                                  MINTOT
00071   004110         04 MONTH-PR             PICTURE XX.                      MINTOT
00072   004120         04 SLASH-1-PR           PICTURE X.                       MINTOT
00073   004130         04 DAY-PR               PICTURE XX.                      MINTOT
00074   004140         04 SLASH-2-PR           PICTURE X.                       MINTOT
00075   004150         04 YEAR-PR              PICTURE XX.                      MINTOT
00076   004160     03 FILLER                   PICTURE X(10).                   MINTOT
00077   004170     03 BRANCH-PR                PICTURE XX.                      MINTOT
00078   004180     03 FILLER                   PICTURE X(17).                   MINTOT
00079   004190     03 SALESMAN-PR              PICTURE XX.                      MINTOT
00080   004200     03 FILLER                   PICTURE X(13).                   MINTOT
00081   005010     03 AMOUNT-PR                PICTURE Z,ZZZ.99.                MINTOT
00082   005020     03 FILLER                   PICTURE X(57).                   MINTOT
00083   005030 01  TITLE.                                                       MINTOT
00084   005040     03 FILLER                   PICTURE X.                       MINTOT
00085   005050     03 DATE-CURRENT             PICTURE X(8) VALUE SPACES.       MINTOT
00086   005060     03 FILLER                   PICTURE X(20) VALUE SPACES.      MINTOT
00087   005070     03 FILLER                   PICTURE X(62) VALUE IS           MINTOT
```

```
00088   005080       'W E E K L Y     S A L E S      A N A L Y S I S     R E P OMINTOT
00089   005090-      ' R T'.                                                    MINTOT
00090   005100     03 FILLER                   PICTURE X(28) VALUE SPACES.      MINTOT
00091   005110     03 FILLER                   PICTURE X(8) VALUE IS 'PAGE NO'. MINTOT
00092   005120     03 PAGE-OUT                 PICTURE ZZ9.                     MINTOT
00093   005130     03 FILLER                   PICTURE X(3) VALUE SPACES.       MINTOT
00094   005140 01  COLUMN-HEADINGS.                                             MINTOT
00095   005150     03 FILLER                   PICTURE X.                       MINTOT
00096   005160     03 FILLER                   PICTURE X(31) VALUE IS SPACES.   MINTOT
00097   005170     03 FILLER         PICTURE X(43) VALUE IS                     MINTOT
00098   005180       'BRANCH          SALESMAN NO          AMOUNT'.             MINTOT
00099   005190     03 FILLER                   PICTURE X(58) VALUE IS SPACES.   MINTOT
00100   005200 01  SALESMAN-TOTAL-LINE.                                         MINTOT
00101   006010     03 FILLER                   PICTURE X.                       MINTOT
00102   006020     03 FILLER                   PICTURE X(65) VALUE IS SPACES.   MINTOT
00103   006030     03 SM-TOTAL-OUT             PICTURE $$$,$$$.99.              MINTOT
00104   006040     03 FILLER                   PICTURE X VALUE '*'.             MINTOT
00105   006050     03 FILLER                   PICTURE X(56) VALUE IS SPACES.   MINTOT
00106   006060 01  BRANCH-TOTAL-LINE.                                           MINTOT
00107   006070     03 FILLER                   PICTURE X.                       MINTOT
00108   006080     03 FILLER                   PICTURE X(81) VALUE IS SPACES.   MINTOT
00109   006090     03 BR-TOTAL-OUT             PICTURE $$$,$$$.99.              MINTOT
00110   006100     03 FILLER                   PICTURE XX VALUE '**'.           MINTOT
00111   006110     03 FILLER                   PICTURE X(39) VALUE IS SPACES.   MINTOT
00112   006120 01  DATE-TOTAL-LINE.                                             MINTOT
00113   006130     03 FILLER                   PICTURE X.                       MINTOT
00114   006140     03 FILLER                   PICTURE X(40) VALUE IS SPACES.   MINTOT
00115   006150     03 FILLER         PICTURE X(23) VALUE IS                     MINTOT
00116   006160       'TOTAL SALES FOR THE DAY'.                                 MINTOT
00117   006170     03 FILLER                   PICTURE X(25) VALUE IS SPACES.   MINTOT
00118   006180     03 D-TOTAL-OUT              PICTURE $$$,$$$.99.              MINTOT
00119   006190     03 FILLER                   PICTURE XXX VALUE IS '***'.      MINTOT
00120   006200     03 FILLER                   PICTURE X(31) VALUE SPACES.      MINTOT
00121   007010 01  FINAL-TOTAL-LINE.                                            MINTOT
00122   007020     03 FILLER                   PICTURE X.                       MINTOT
00123   007030     03 FILLER                   PICTURE X(85) VALUE SPACES.      MINTOT
00124   007040     03 FILLER                   PICTURE X(15) VALUE IS           MINTOT
00125   007050       'FINAL TOTAL    '.                                         MINTOT
00126   007060     03 F-TOTAL-OUT              PICTURE $$$,$$$.99.              MINTOT
00127   007070     03 FILLER                   PICTURE X(4) VALUE IS '****'.    MINTOT
00128   007080     03 FILLER                   PICTURE X(18) VALUE SPACES.      MINTOT
00129   007090 PROCEDURE DIVISION.                                              MINTOT
00130   007100 010-OPEN.                                                        MINTOT
00131   007110     OPEN INPUT SALESMAN-FILE                                     MINTOT
00132   007120          OUTPUT REPORT-FILE.                                     MINTOT
00133   007130                                                                  MINTOT
00134   007140     NOTE ******************************************************* MINTOT
00135   007150     *                                                          * MINTOT
00136   007160     * THIS ROUTINE CALLS AN ASSEMBLER PROGRAM WHICH EXTRACTS   * MINTOT
00137   007170     * THE CURRENT DATE FROM THE COMMUNICATION REGION           * MINTOT
00138   007180     *                                                          * MINTOT
00139   007190     ************************************************************.MINTOT
00140   007200                                                                  MINTOT
00141   008010     CALL 'DATEROT' USING DATE-CURRENT.                           MINTOT
00142   008020     PERFORM 130-TITLE-ROUTINE THRU 130-EXIT.                     MINTOT
00143   008030 020-BEGIN-JOB.                                                   MINTOT
00144   008040     READ SALESMAN-FILE INTO SCARD AT END GO TO 140-END-OF-JOB.   MINTOT
00145   008050     MOVE SPACES TO DETAIL-LINE.                                  MINTOT
00146   008060     IF DATE-CD IS NOT = DATE-HOLD                                MINTOT
00147   008070         PERFORM 040-SWITCH-1 THRU 040-EXIT                       MINTOT
00148   008080         GO TO 030-MAIN-ROUTINE.                                  MINTOT
00149   008090     IF BRANCH-CD IS NOT = BRANCH-HOLD                            MINTOT
00150   008100         PERFORM 070-BRANCH-ROUTINE THRU 070-EXIT,                MINTOT
00151   008110         GO TO 030-MAIN-ROUTINE.                                  MINTOT
00152   008120     IF SALESMAN-CD NOT = SALESMAN-HOLD                           MINTOT
00153   008130         PERFORM 090-SALESMAN-ROUTINE THRU 090-EXIT.              MINTOT
00154   008140 030-MAIN-ROUTINE.                                                MINTOT
00155   008150     ADD AMOUNT-CD TO SALESMAN-TOTAL.                             MINTOT
00156   008160     MOVE BRANCH-CD TO BRANCH-PR.                                 MINTOT
00157   008170     MOVE SALESMAN-CD TO SALESMAN-PR.                             MINTOT
00158   008180     MOVE AMOUNT-CD TO AMOUNT-PR.                                 MINTOT
00159   008190     WRITE OUTPUT-LINE FROM DETAIL-LINE AFTER ADVANCING VARIABLE  MINTOT
00160   008200         LINES.                                                   MINTOT
00161   009010     ADD 1 TO LINE-COUNT.                                         MINTOT
00162   009020     MOVE SINGLE TO VARIABLE.                                     MINTOT
00163   009030     IF LINE-COUNT > 48 PERFORM 130-TITLE-ROUTINE THRU 130-EXIT.  MINTOT
00164   009040     GO TO 020-BEGIN-JOB.                                         MINTOT
00165   009050                                                                  MINTOT
00166   009060     NOTE ******************************************************* MINTOT
00167   009070     *                                                          * MINTOT
00168   009080     * THIS ROUTINE IS ENTERED WHEN PROCESSING THE FIRST CARD   * MINTOT
00169   009090     *                                                          * MINTOT
00170   009100     ************************************************************.MINTOT
00171   009110                                                                  MINTOT
00172   009120 040-SWITCH-1.                                                    MINTOT
00173   009130     IF SWITCH = '0'                                              MINTOT
00174   009140         MOVE '1' TO SWITCH                                       MINTOT
```

```
009150         MOVE BRANCH-CD TO BRANCH-HOLD                             MINTOT
009160         MOVE SALESMAN-CD TO SALESMAN-HOLD                         MINTOT
009170         GO TO 060-END-DATE-ROUTINE.                               MINTOT
009180                                                                   MINTOT
009190     NOTE ******************************************************** MINTOT
009200     *                                                           * MINTOT
010010     * THIS ROUTINE IS ENTERED WHEN THERE IS A CHANGE IN DATE   * MINTOT
010020     *                                                           * MINTOT
010030     *************************************************************.MINTOT
010040                                                                   MINTOT
010050 050-DATE-ROUTINE.                                                 MINTOT
010060     PERFORM 070-BRANCH-ROUTINE THRU 070-EXIT.                     MINTOT
010070     ADD DATE-TOTAL TO FINAL-TOTAL.                                MINTOT
010080     MOVE DATE-TOTAL TO D-TOTAL-OUT.                               MINTOT
010090     WRITE OUTPUT-LINE FROM DATE-TOTAL-LINE AFTER ADVANCING 2      MINTOT
010100         LINES.                                                    MINTOT
010110     ADD 2 TO LINE-COUNT.                                          MINTOT
010120     MOVE ZERO TO DATE-TOTAL.                                      MINTOT
010130 060-END-DATE-ROUTINE.                                             MINTOT
010140     MOVE DATE-CD TO DATE-HOLD.                                    MINTOT
010150     MOVE MONTH-CD TO MONTH-PR.                                    MINTOT
010160     MOVE '/' TO SLASH-1-PR, SLASH-2-PR.                           MINTOT
010170     MOVE DAY-CD TO DAY-PR.                                        MINTOT
010180     MOVE YEAR-CD TO YEAR-PR.                                      MINTOT
010190 040-EXIT.  EXIT.                                                  MINTOT
010200                                                                   MINTOT
011010     NOTE ******************************************************** MINTOT
011020     *                                                           * MINTOT
011030     * THIS ROUTINE IS ENTERED WHEN THERE IS A CHANGE IN BRANCH * MINTOT
011040     *                                                           * MINTOT
011050     *************************************************************.MINTOT
011060                                                                   MINTOT
011070 070-BRANCH-ROUTINE.                                               MINTOT
011080     PERFORM 090-SALESMAN-ROUTINE THRU 090-EXIT.                   MINTOT
011090     ADD BRANCH-TOTAL TO DATE-TOTAL.                               MINTOT
011100     MOVE BRANCH-TOTAL TO BR-TOTAL-OUT.                            MINTOT
011110     WRITE OUTPUT-LINE FROM BRANCH-TOTAL-LINE AFTER ADVANCING      MINTOT
011120       2 LINES.                                                    MINTOT
011130     ADD 2 TO LINE-COUNT.                                          MINTOT
011140     MOVE ZERO TO BRANCH-TOTAL.                                    MINTOT
011150     MOVE BRANCH-CD TO BRANCH-HOLD.                                MINTOT
011160 070-EXIT.  EXIT.                                                  MINTOT
011170                                                                   MINTOT
011180     NOTE ******************************************************** MINTOT
011190     *                                                           * MINTOT
011200     * THIS ROUTINE IS ENTERED                                   * MINTOT
012010     * WHEN THERE IS A CHANGE IN SALESMAN                        * MINTOT
012020     *                                                           * MINTOT
012030     *************************************************************.MINTOT
012040                                                                   MINTOT
012050 090-SALESMAN-ROUTINE.                                             MINTOT
012060     ADD SALESMAN-TOTAL TO BRANCH-TOTAL.                           MINTOT
012070     MOVE SALESMAN-TOTAL TO SM-TOTAL-OUT.                          MINTOT
012080     WRITE OUTPUT-LINE FROM SALESMAN-TOTAL-LINE AFTER ADVANCING    MINTOT
012090         2 LINES.                                                  MINTOT
012100     MOVE DOUBLE TO VARIABLE.                                      MINTOT
012110     ADD 3 TO LINE-COUNT.                                          MINTOT
012120     MOVE ZERO TO SALESMAN-TOTAL.                                  MINTOT
012130                                                                   MINTOT
012140     MOVE SALESMAN-CD TO SALESMAN-HOLD.                            MINTOT
012150 090-EXIT. EXIT.                                                   MINTOT
012160                                                                   MINTOT
012170     NOTE ******************************************************** MINTOT
012180     *                                                           * MINTOT
012190     * THIS ROUTINE IS ENTERED TO PRINT THE HEADINGS             * MINTOT
012200     * ON THE REPORT                                             * MINTOT
013010     *                                                           * MINTOT
013020     *************************************************************.MINTOT
013030                                                                   MINTOT
013040 130-TITLE-ROUTINE.                                                MINTOT
013050     MOVE PAGE-NO TO PAGE-OUT.                                     MINTOT
013060     WRITE OUTPUT-LINE FROM TITLE AFTER ADVANCING                  MINTOT
013070         TO-TOP-OF-PAGE.                                           MINTOT
013080     MOVE DOUBLE TO VARIABLE.                                      MINTOT
013090     WRITE OUTPUT-LINE FROM COLUMN-HEADINGS AFTER ADVANCING        MINTOT
013100         2 LINES.                                                  MINTOT
013110     ADD 1 TO PAGE-NO.                                             MINTOT
013120     MOVE 4 TO LINE-COUNT.                                         MINTOT
013130 130-EXIT.  EXIT.                                                  MINTOT
013140                                                                   MINTOT
013150     NOTE ******************************************************** MINTOT
013160     *                                                           * MINTOT
013170     * THIS ROUTINE IS ENTERED AFTER THE LAST CARD HAS BEEN      * MINTOT
013180     * READ   IT WRITES ALL OF THE INTERMEDIATE TOTALS FOR       * MINTOT
013190     * THE LAST SALESMAN PROCESSED, THEN WRITES THE FINAL        * MINTOT
013200     * TOTALS AND CLOSES THE FILES                               * MINTOT
014010     *                                                           * MINTOT
014020     *************************************************************.MINTOT
014030                                                                   MINTOT
014040 140-END-OF-JOB.                                                   MINTOT
014050     PERFORM 050-DATE-ROUTINE.                                     MINTOT
014060     MOVE FINAL-TOTAL TO F-TOTAL-OUT.                              MINTOT
014070     WRITE OUTPUT-LINE FROM FINAL-TOTAL-LINE AFTER ADVANCING       MINTOT
014080         3 LINES.                                                  MINTOT
014090     CLOSE SALESMAN-FILE REPORT-FILE.                              MINTOT
014100     STOP RUN.                                                     MINTOT
```

PROGRAMMING ASSIGNMENT 1

INTERMEDIATE, MINOR TOTALS

INSTRUCTIONS

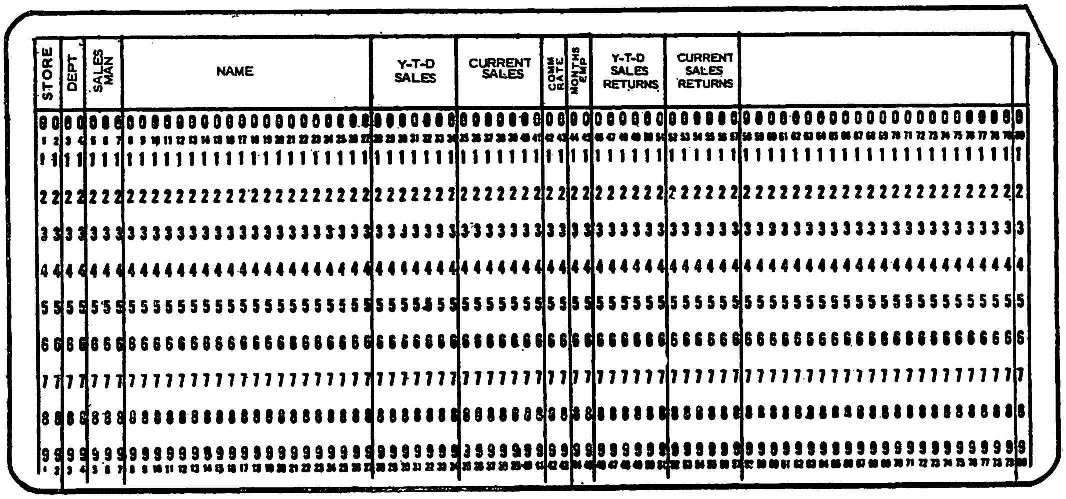
From the Sales card illustrated below, a Sales Analysis Report is to be prepared. An intermediate total is to be taken when there is a change in Store Number, a minor total when there is a change in Department Number.

Report and column headings are to be printed on the report. Print the Current Date at the top of the report. Extract the date from the area in the supervisory program containing the date.

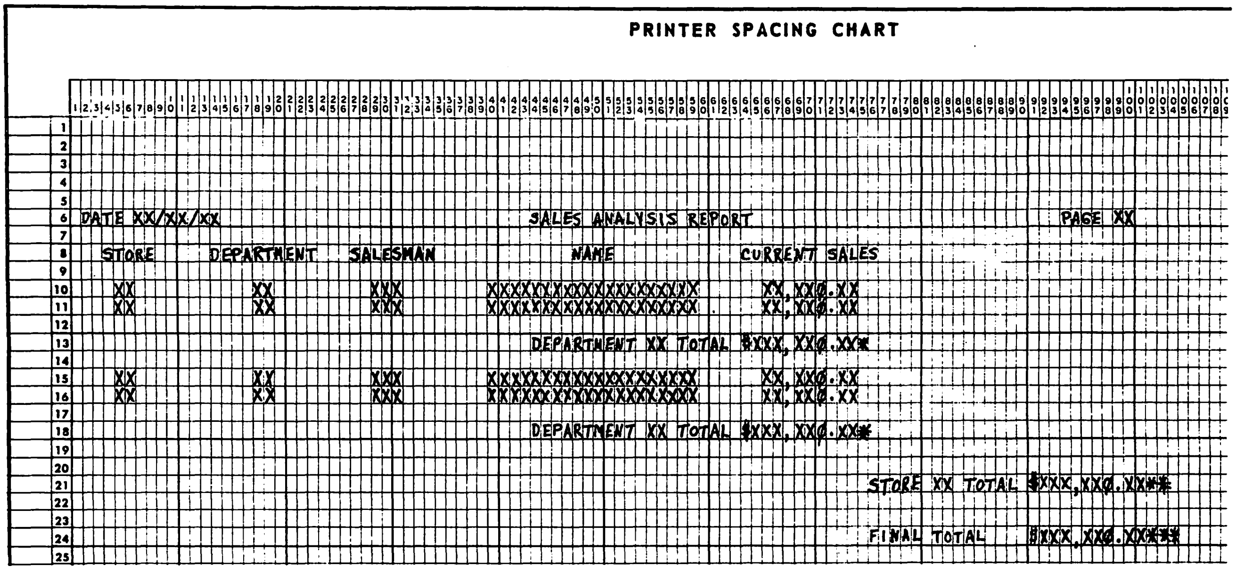
Cards are to be sorted by Salesman Number, Department Number, Store Number.

Test data is contained in the appendices.

INPUT: Sales Cards



OUTPUT: Sales Analysis Report


PRINTER SPACING CHART

```
DATE XX/XX/XX                           SALES ANALYSIS REPORT                         PAGE XX

   STORE    DEPARTMENT  SALESMAN            NAME          CURRENT SALES

    XX          XX        XXX      XXXXXXXXXXXXXXXXXXXX    XX,XX0.XX
    XX          XX        XXX      XXXXXXXXXXXXXXXXXXXX    XX,XX0.XX

                                       DEPARTMENT XX TOTAL $XXX,XX0.XX*

    XX          XX        XXX      XXXXXXXXXXXXXXXXXXXX    XX,XX0.XX
    XX          XX        XXX      XXXXXXXXXXXXXXXXXXXX    XX,XX0.XX

                                       DEPARTMENT XX TOTAL $XXX,XX0.XX*


                                                                     STORE XX TOTAL $XXX,XX0.XX**


                                                                     FINAL TOTAL    $XXX,XX0.XX***
```

CHAPTER 10

PROGRAMMING ASSIGNMENT 2

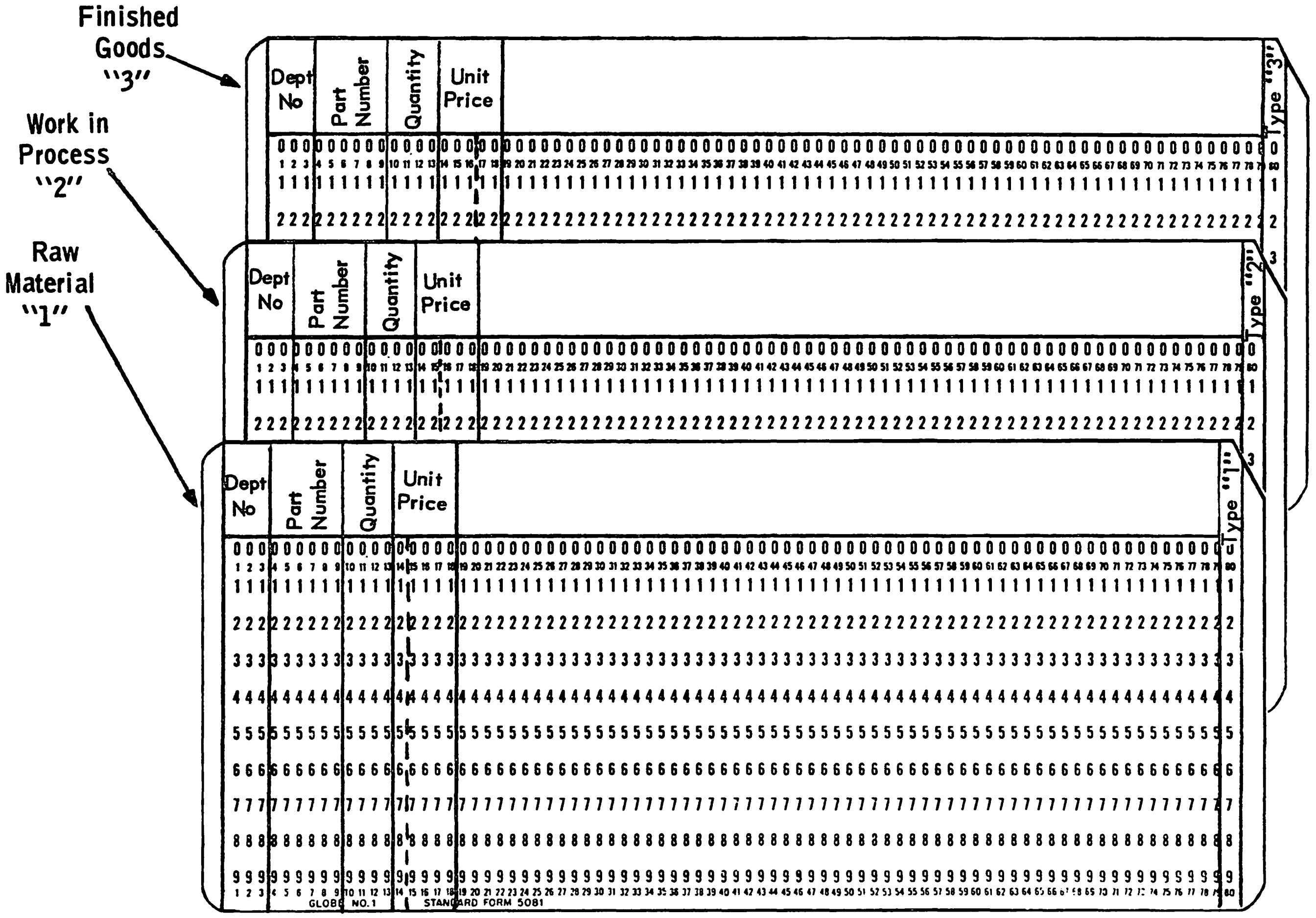
INSTRUCTIONS

Write the COBOL program to process Physical Inventory records and create a Physical Inventory Report. A Physical Inventory consists of a count of the number of parts a company currently has. These parts may be in one of three conditions: Raw Material, Work in Process, or Finished Goods. When the part consists of merely Raw Material, then no work has been done to build the part. When a part is in Work in Process it is in the process of being built. Finished Goods means that the part has been completely built and is ready to be sold to a customer.

The test data for the program is shown on page 10.21.

INPUT: Physical Inventory Cards

The Physical Inventory cards consist of three types – one for the Raw Material count, one for the Work in Process count, and one for the Finished Goods count. The format of these cards is illustrated below.

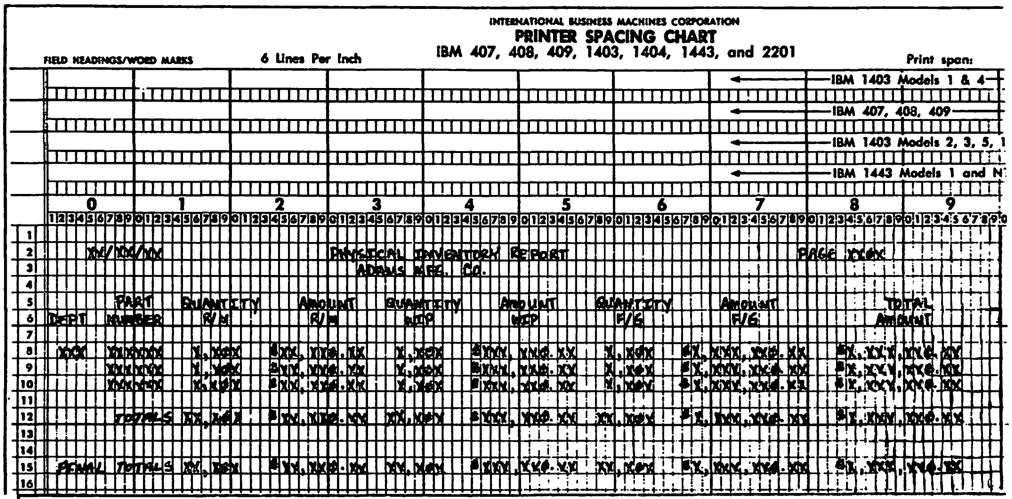

Finished Goods "3"
Work in Process "2"
Raw Material "1"
Dept No
Part Number
Quantity
Unit Price
Type "3"
Type "2"
Type "1"
GLOBE NO. 1
STANDARD FORM 5081

Note from the card formats that each contains a type code in column 80 to identify the type of card being processed. Note also that the number of decimal places to the right of the decimal point in the unit prices is different dependent upon the type of record being processed. For Raw Material, the unit price has the format X.XXXX, for Work in Process the format is XX.XXX, and for Finished Goods, the format is XXX.XX.

OUTPUT: Physical Inventory Report

There is one card per part number for each type. Thus, there would be one raw material card for part number 996543, one Work in Process card for part number 996543, and one Finished Goods card for this part number. Note, however, that there need not be all three cards, that is, one part number may only have a card for Finished Goods.

The report is to have the department number group indicated and the part number group printed. The print line is to have the total amount for each of the three categories. The total amount is determined by multiplying the unit price in the card input by the quantity in the card input. Note that all values on the report are specified as dollars and cents.


INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, and 2201
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
Print span:
IBM 1403 Models 1 & 4
IBM 407, 408, 409
IBM 1403 Models 2, 3, 5, 1
IBM 1443 Models 1 and N
XX/XX/XX
PHYSICAL INVENTORY REPORT
ADAMS MFG. CO.
PAGE XXXX
DEPT
PART NUMBER
QUANTITY R/M
AMOUNT R/M
QUANTITY WIP
AMOUNT WIP
QUANTITY F/G
AMOUNT F/G
TOTAL AMOUNT
TOTALS
FINAL TOTALS

```
     10/16/72                       PHYSICAL  INVENTORY  REPORT                       PAGE    1
                                         ADAMS MFG. CO.

          PART    QUANTITY        AMOUNT    QUANTITY       AMOUNT    QUANTITY        AMOUNT              TOTAL
  DEPT   NUMBER     R/M             R/M        WIP           WIP        F/G            F/G               AMOUNT

   25.0  279816      400      $    600.00     1,000    $ 20,500.00     2,000    $  300,000.00      $  321,100.00
         218760        0      $       .00     7,655    $ 76,550.00        90    $      900.00      $   77,450.00
         776521        0      $       .00         0    $       .00     9,005    $   11,256.25      $   11,256.25

          TOTALS     400      $    600.00     8,655    $ 97,050.00    11,095    $  312,156.25      $  409,806.25


   FINAL  TOTALS  23,724      $56,723.40     18,781    $727,299.40    73,284    $1,972,388.56      $2,756,411.36
```

TEST DATA – ASSIGNMENT NO. 2

Dept No (col 1-3)	Part Number (col 4-9)	Quantity (col 10-13)	Unit Price (col 14-18)	Type (col 80)
250	279816	0400	15000	1
250	279816	1000	20500	2
250	279816	2000	15000	3
250	218760	7655	10000	2
250	218760	0090	01000	3
250	776521	9005	00125	3
543	887656	0095	45327	1
543	667656	8896	00987	3
546	667656	0096	66540	2
668	776549	8600	00675	3

CHAPTER 11

TABLE LOOKUP

INTRODUCTION

In many business programming applications it is desirable to organize Reference Data in the form of a "table", store the table in core, and retrieve those portions of the table that are needed in the solution of a problem by means of the COBOL program. Tables can be of many types including: tables of wage rates, tables of commissions, tables of items descriptions, tax tables, etc.

For example, in a payroll application if there are 10 different Pay Rates, these Pay Rates could be set up in a table. Each of these rates could be assigned a 2 digit Pay Rate Code from 01 to 10. From the Pay Rate Code the actual Pay Rate could be "looked up" in the table. See the illustration below.

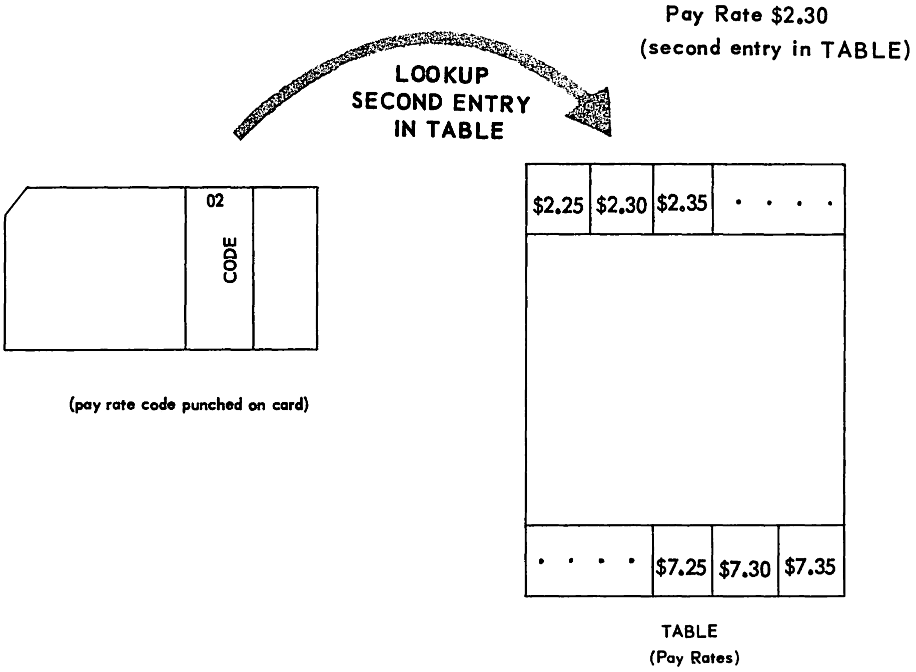


Figure 11-1 Example of Table Lookup

In the example above the Master Card contains a Pay Rate Code of "02". When the COBOL program is executed, the second "element" or entry in the table is retrieved ($2.30).

The use of tables has several advantages. For example, if a company has 5,000 employees and each employee has one of ten different pay rates punched on his individual master card, a change in pay for all employees would result in a change to all 5,000 master cards. If the Pay Rates were stored in the form of a table, only the 10 entries in the table would require changing.

The use of tables can also save card columns. For example, tables of descriptions can be set up and referenced by a digit code on the input data card. The example developed in this chapter illustrates an application of this type.

TABLE LOOKUP - EXAMPLE 1

To illustrate the concepts of table lookup, this Chapter develops a problem in which Input Data Cards consist of Master Student Cards which contain the Student Number Field, a Name Field, and a Code Field. The Code Field contains a 2 digit number from 01 to 10 which identifies the Student's Major in College according to the following Chart.

01	**Business Administration**
02	**Data Processing**
03	**English**
04	**History**
05	**Journalism**
06	**Music**
07	**Physical Education**
08	**Political Science**
09	**Social Science**
10	**Zoology**

The following is an illustration of the Master Student Card.

Input

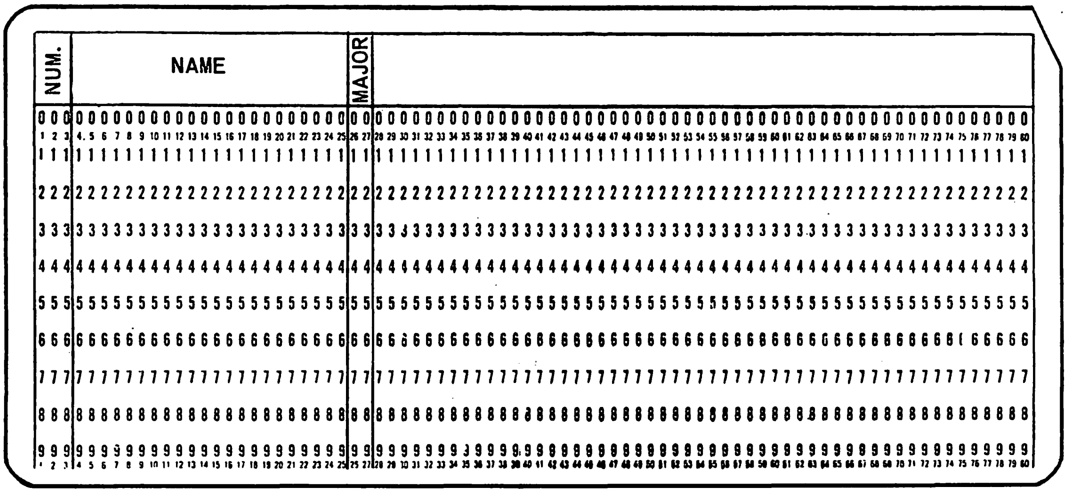
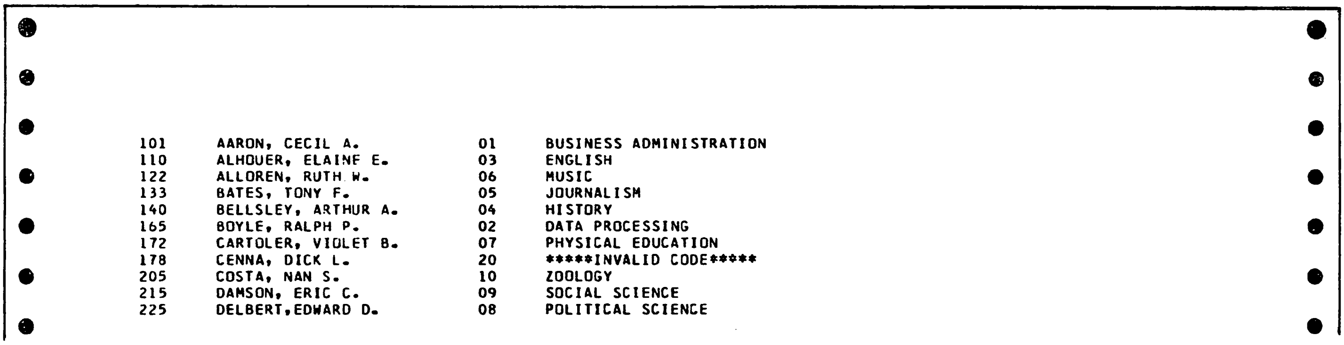


Figure 11-2 Input Data Card

Output is to consist of a printed report listing the Student Number, the Student Name, the Code, and the Major. The Major is to be "looked up" in a table set up within the COBOL Program. The following is an illustration of the output that is to be produced.

Output



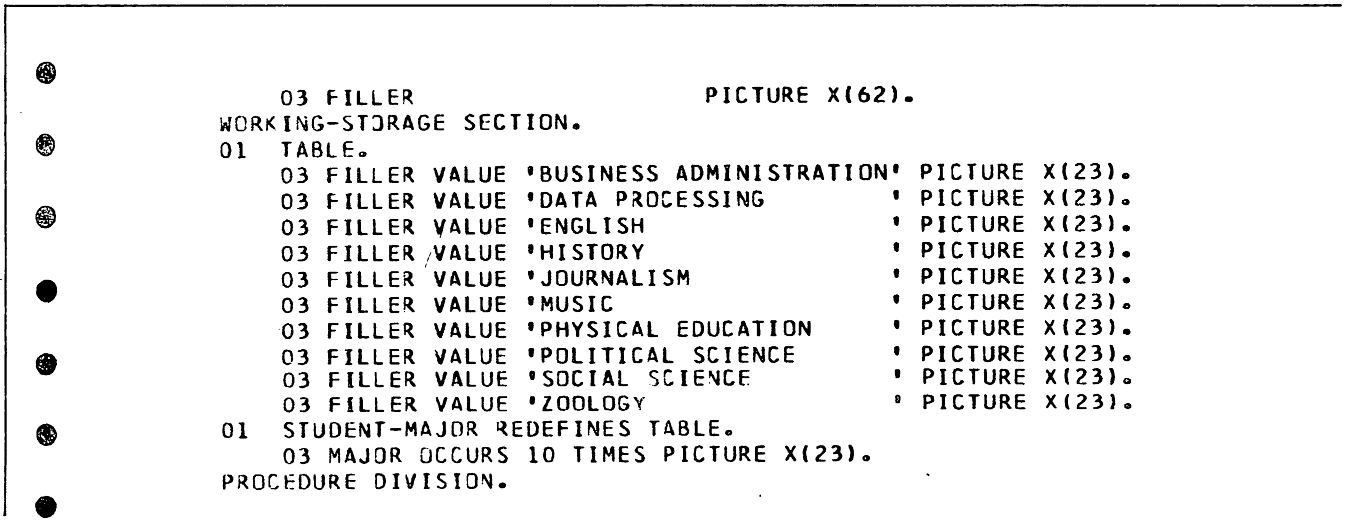
```
101     AARON, CECIL A.              01     BUSINESS ADMINISTRATION
110     ALHOUER, ELAINF E.           03     ENGLISH
122     ALLOREN, RUTH W.             06     MUSIC
133     BATES, TONY F.               05     JOURNALISM
140     BELLSLEY, ARTHUR A.          04     HISTORY
165     BOYLE, RALPH P.              02     DATA PROCESSING
172     CARTOLER, VIOLET B.          07     PHYSICAL EDUCATION
178     CENNA, DICK L.               20     *****INVALID CODE*****
205     COSTA, NAN S.                10     ZOOLOGY
215     DAMSON, ERIC C.              09     SOCIAL SCIENCE
225     DELBERT,EDWARD D.            08     POLITICAL SCIENCE
```

Figure 11-3 Sample Output

In applications in which there are few entries within the table and where the entries do not change frequently, the table is normally developed within the COBOL Program. In applications where there are many entries in the table, where there are frequent changes to the table, or when the table is referenced by other programs, it is desirable to construct a separate file of cards for the table, or to store the table on a direct access device such as a disk storage unit.

To set up the table in core and within the COBOL Program, it is necessary to define the table using a Level 01 entry in WORKING-STORAGE, and to further define the elements within the table. The following is an illustration of a portion of the WORKING-STORAGE SECTION which sets up the table.

WORKING-STORAGE SECTION



```
           03 FILLER                      PICTURE X(62).
       WORKING-STORAGE SECTION.
       01  TABLE.
           03 FILLER VALUE 'BUSINESS ADMINISTRATION' PICTURE X(23).
           03 FILLER VALUE 'DATA PROCESSING        ' PICTURE X(23).
           03 FILLER VALUE 'ENGLISH                ' PICTURE X(23).
           03 FILLER VALUE 'HISTORY                ' PICTURE X(23).
           03 FILLER VALUE 'JOURNALISM             ' PICTURE X(23).
           03 FILLER VALUE 'MUSIC                  ' PICTURE X(23).
           03 FILLER VALUE 'PHYSICAL EDUCATION     ' PICTURE X(23).
           03 FILLER VALUE 'POLITICAL SCIENCE      ' PICTURE X(23).
           03 FILLER VALUE 'SOCIAL SCIENCE         ' PICTURE X(23).
           03 FILLER VALUE 'ZOOLOGY                ' PICTURE X(23).
       01  STUDENT-MAJOR REDEFINES TABLE.
           03 MAJOR OCCURS 10 TIMES PICTURE X(23).
       PROCEDURE DIVISION.
```

Figure 11-4 WORKING-STORAGE SECTION to describe a Table

SUBSCRIPTS

Reference is made to the elements of the Table by means of a "subscript". A "subscript" is a numeric value, or a data-name referencing a numeric value, which is placed in parenthesis following the data item referenced. For example, to extract the second entry from the table, the following statement could be written.

```
MOVE MAJOR (2) TO MAJOR-PR.
```

Figure 11-5 MOVE Statement using a Subscript

The effect of the statement above is to extract the second element from the Table (DATA PROCESSING) and to move this element to the area referenced by MAJOR-PR.

OCCURS CLAUSE

To reference elements in a table in this manner requires that the "Table" be REDEFINED. When a Table is redefined for subscripting purposes, individual elements are not named. Instead, the OCCURS Clause is used to specify how many times the elements within the Table occur.

The OCCURS Clause is used in defining related sets of repeated data, such as tables, lists, matrices, etc. and is used in this example to specify the number of times that a data item with the same format is repeated. The effect of the OCCURS Clause illustrated in Figure 11-4 is to specify the number of elements in the Table so that each element may be referenced in the Procedure Division by subscripting.

The format of the OCCURS Clause is illustrated below.

```
OCCURS integer-2 TIMES

     { ASCENDING  }
   [ {            } KEY IS data-name-2 [data-name-3] ... ] ...
     { DESCENDING }

     [INDEXED BY index-name-1[index-name-2] ... ]
```

Figure 11-6 OCCURS Clause

Items described by an OCCURS clause can be referenced only by subscripting. Subscripting enables an item to be referenced by its position in a table or list rather than by individual name. The OCCURS clause must not be used in any Record Description entry having a level number 01 or 88.

It should be noted that the optional entries in the OCCURS clause will be discussed in Chapter 12.

The subscript need not be stated directly in numeric form. The subscript can be the name of a data item in the program. When reference is made to the subscript when the program is being executed, the current value of the data item is substituted as the value of the subscript. This technique allows more flexibility in subscripting as it permits the value represented by the subscript to be altered.

The following example illustrates a MOVE Statement in the Procedure Division, which would cause an element from the table to be moved to the area on the output line referenced by MAJOR-PR.

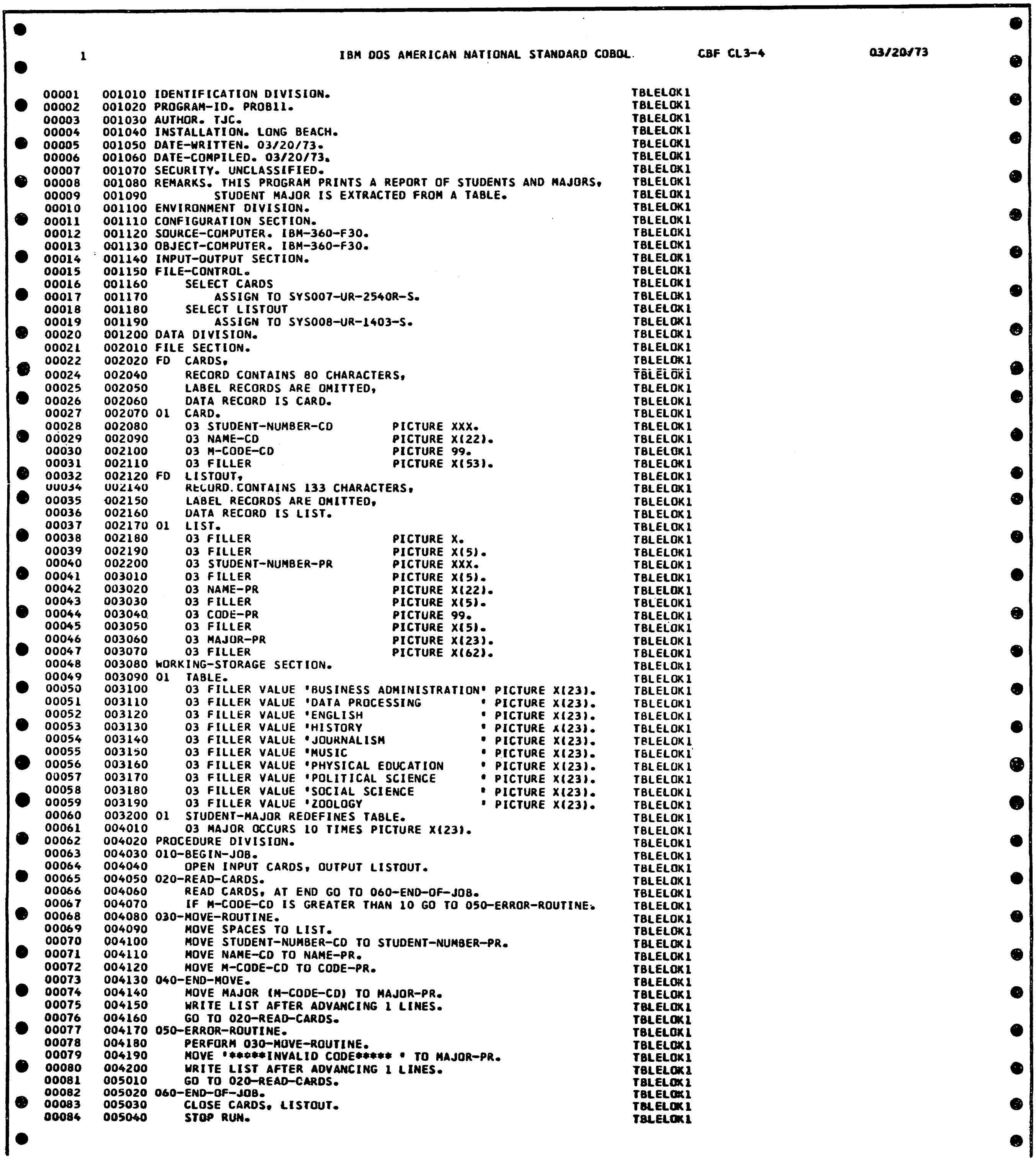
```
    MOVE MAJOR (M-CODE-CD) TO MAJOR-PR.
```

Figure 11-7 MOVE Statement referencing an element in a table

In the sample problem, if a card is read in which the Code field (M-CODE-CD) contains a value of 01, the first element from the Table would be extracted (BUSINESS ADMINISTRATION). If a card is read with which contains a Code Field with the value of 02, the second element from the Table is extracted, etc.

The program to perform the table lookup operation in which the Student's major is extracted from a table is illustrated on the following page. The input and output are illustrated in Figure 11-2 and Figure 11-3.

COBOL PROGRAM



```
      1                                IBM DOS AMERICAN NATIONAL STANDARD COBOL        CBF CL3-4            03/20/73

00001   001010 IDENTIFICATION DIVISION.                                         TBLELOK1
00002   001020 PROGRAM-ID. PROB11.                                              TBLELOK1
00003   001030 AUTHOR. TJC.                                                     TBLELOK1
00004   001040 INSTALLATION. LONG BEACH.                                        TBLELOK1
00005   001050 DATE-WRITTEN. 03/20/73.                                          TBLELOK1
00006   001060 DATE-COMPILED. 03/20/73.                                         TBLELOK1
00007   001070 SECURITY. UNCLASSIFIED.                                          TBLELOK1
00008   001080 REMARKS. THIS PROGRAM PRINTS A REPORT OF STUDENTS AND MAJORS,    TBLELOK1
00009   001090         STUDENT MAJOR IS EXTRACTED FROM A TABLE.                 TBLELOK1
00010   001100 ENVIRONMENT DIVISION.                                            TBLELOK1
00011   001110 CONFIGURATION SECTION.                                           TBLELOK1
00012   001120 SOURCE-COMPUTER. IBM-360-F30.                                    TBLELOK1
00013   001130 OBJECT-COMPUTER. IBM-360-F30.                                    TBLELOK1
00014   001140 INPUT-OUTPUT SECTION.                                            TBLELOK1
00015   001150 FILE-CONTROL.                                                    TBLELOK1
00016   001160     SELECT CARDS                                                 TBLELOK1
00017   001170         ASSIGN TO SYS007-UR-2540R-S.                             TBLELOK1
00018   001180     SELECT LISTOUT                                               TBLELOK1
00019   001190         ASSIGN TO SYS008-UR-1403-S.                              TBLELOK1
00020   001200 DATA DIVISION.                                                   TBLELOK1
00021   002010 FILE SECTION.                                                    TBLELOK1
00022   002020 FD  CARDS,                                                       TBLELOK1
00024   002040     RECORD CONTAINS 80 CHARACTERS,                               TBLELOK1
00025   002050     LABEL RECORDS ARE OMITTED,                                   TBLELOK1
00026   002060     DATA RECORD IS CARD.                                         TBLELOK1
00027   002070 01  CARD.                                                        TBLELOK1
00028   002080     03 STUDENT-NUMBER-CD         PICTURE XXX.                    TBLELOK1
00029   002090     03 NAME-CD                   PICTURE X(22).                  TBLELOK1
00030   002100     03 M-CODE-CD                 PICTURE 99.                     TBLELOK1
00031   002110     03 FILLER                    PICTURE X(53).                  TBLELOK1
00032   002120 FD  LISTOUT,                                                     TBLELOK1
00034   002140     RECORD CONTAINS 133 CHARACTERS,                              TBLELOK1
00035   002150     LABEL RECORDS ARE OMITTED,                                   TBLELOK1
00036   002160     DATA RECORD IS LIST.                                         TBLELOK1
00037   002170 01  LIST.                                                        TBLELOK1
00038   002180     03 FILLER                    PICTURE X.                      TBLELOK1
00039   002190     03 FILLER                    PICTURE X(5).                   TBLELOK1
00040   002200     03 STUDENT-NUMBER-PR         PICTURE XXX.                    TBLELOK1
00041   003010     03 FILLER                    PICTURE X(5).                   TBLELOK1
00042   003020     03 NAME-PR                   PICTURE X(22).                  TBLELOK1
00043   003030     03 FILLER                    PICTURE X(5).                   TBLELOK1
00044   003040     03 CODE-PR                   PICTURE 99.                     TBLELOK1
00045   003050     03 FILLER                    PICTURE X(5).                   TBLELOK1
00046   003060     03 MAJOR-PR                  PICTURE X(23).                  TBLELOK1
00047   003070     03 FILLER                    PICTURE X(62).                  TBLELOK1
00048   003080 WORKING-STORAGE SECTION.                                         TBLELOK1
00049   003090 01  TABLE.                                                       TBLELOK1
00050   003100     03 FILLER VALUE 'BUSINESS ADMINISTRATION' PICTURE X(23).     TBLELOK1
00051   003110     03 FILLER VALUE 'DATA PROCESSING        ' PICTURE X(23).     TBLELOK1
00052   003120     03 FILLER VALUE 'ENGLISH                ' PICTURE X(23).     TBLELOK1
00053   003130     03 FILLER VALUE 'HISTORY                ' PICTURE X(23).     TBLELOK1
00054   003140     03 FILLER VALUE 'JOURNALISM             ' PICTURE X(23).     TBLELOK1
00055   003150     03 FILLER VALUE 'MUSIC                  ' PICTURE X(23).     TBLELOK1
00056   003160     03 FILLER VALUE 'PHYSICAL EDUCATION     ' PICTURE X(23).     TBLELOK1
00057   003170     03 FILLER VALUE 'POLITICAL SCIENCE      ' PICTURE X(23).     TBLELOK1
00058   003180     03 FILLER VALUE 'SOCIAL SCIENCE         ' PICTURE X(23).     TBLELOK1
00059   003190     03 FILLER VALUE 'ZOOLOGY                ' PICTURE X(23).     TBLELOK1
00060   003200 01  STUDENT-MAJOR REDEFINES TABLE.                               TBLELOK1
00061   004010     03 MAJOR OCCURS 10 TIMES PICTURE X(23).                      TBLELOK1
00062   004020 PROCEDURE DIVISION.                                              TBLELOK1
00063   004030 010-BEGIN-JOB.                                                   TBLELOK1
00064   004040     OPEN INPUT CARDS, OUTPUT LISTOUT.                            TBLELOK1
00065   004050 020-READ-CARDS.                                                  TBLELOK1
00066   004060     READ CARDS, AT END GO TO 060-END-OF-JOB.                     TBLELOK1
00067   004070     IF M-CODE-CD IS GREATER THAN 10 GO TO 050-ERROR-ROUTINE.     TBLELOK1
00068   004080 030-MOVE-ROUTINE.                                                TBLELOK1
00069   004090     MOVE SPACES TO LIST.                                         TBLELOK1
00070   004100     MOVE STUDENT-NUMBER-CD TO STUDENT-NUMBER-PR.                 TBLELOK1
00071   004110     MOVE NAME-CD TO NAME-PR.                                     TBLELOK1
00072   004120     MOVE M-CODE-CD TO CODE-PR.                                   TBLELOK1
00073   004130 040-END-MOVE.                                                    TBLELOK1
00074   004140     MOVE MAJOR (M-CODE-CD) TO MAJOR-PR.                          TBLELOK1
00075   004150     WRITE LIST AFTER ADVANCING 1 LINES.                          TBLELOK1
00076   004160     GO TO 020-READ-CARDS.                                        TBLELOK1
00077   004170 050-ERROR-ROUTINE.                                               TBLELOK1
00078   004180     PERFORM 030-MOVE-ROUTINE.                                    TBLELOK1
00079   004190     MOVE '*****INVALID CODE***** ' TO MAJOR-PR.                  TBLELOK1
00080   004200     WRITE LIST AFTER ADVANCING 1 LINES.                          TBLELOK1
00081   005010     GO TO 020-READ-CARDS.                                        TBLELOK1
00082   005020 060-END-OF-JOB.                                                  TBLELOK1
00083   005030     CLOSE CARDS, LISTOUT.                                        TBLELOK1
00084   005040     STOP RUN.                                                    TBLELOK1
```

Figure 11-8 COBOL Program

TABLE LOOKUP - EXAMPLE 2

Table lookup operations can take many forms. In the previous example of a table lookup operation a table with 10 individual elements within that table was illustrated. In some types of applications it is desirable to be able to handle multi-level tables, that is, a form of table in which two reference points are required to locate a specific element within the table. Road maps frequently contain "multi-level" tables. On a typical road map a table of distances between two cities is presented. To find the distance between any two cities, the names of the cities are located on the table and where the vertical and horizontal lines representing these cities intersect the distance is given.

Multi-Level Table

TO FROM	BAKERSFIELD, CAL	DALLES (THE), ORE	EUREKA, CAL	FRESNO, CAL	LAKE TAHOE, CAL	LOS ANGELES, CAL	MONTEREY, CAL	OAKLAND, CAL	PORTLAND, ORE	RED BLUFF, CAL	SACRAMENTO, CAL	SAN BERNARDINO, CAL	SAN DIEGO, CAL	SAN FRANCISCO, CAL	SAN JOSE, CAL	SANTA BARBARA, CAL	SEATTLE, WASH	SPOKANE, WASH	VANCOUVER, B. C.	YOSEMITE (VIA MERCED)
ALTURAS, CAL	586	382	307	479	224	652	482	370	434	178	310	629	755	374	413	698	626	596	772	508
ASTORIA, ORE	1002	188	504	895	764	1116	898	786	106	594	726	1166	1242	790	829	1112	205	476	351	924
BAKERSFIELD, CAL		892	568	107	379	114	229	283	920	408	276	164	240	287	254	150	1112	1153	1258	248
BARSTOW, CAL	137	972	705	244	419	134	366	420	1057	545	413	78	204	424	391	204	1226	1186	1332	385
BISHOP, CAL	214	759	574	247	206	275	339	294	811	383	280	252	378	298	292	320	1003	973	1119	155
BLYTHE, CAL	346	1193	913	452	640	239	575	629	1255	754	622	182	240	633	600	333	1447	1370	1553	594
CHICO, CAL	371	523	230	264	150	485	288	173	551	39	95	535	611	177	215	492	743	784	883	293
CRESCENT CITY, CAL	654	438	86	547	435	768	494	384	367	246	378	818	894	374	425	708	559	710	705	576
DALLES (THE), ORE	892		527	785	606	1006	788	676	88	484	616	1011	1132	680	719	1002	268	286	381	814
EL CENTRO, CAL	325	1172	892	432	619	219	554	608	1245	733	601	161	123	612	579	312	1416	1386	1532	573
EUREKA, CAL	568	527		460	380	682	408	298	456	191	301	732	808	288	339	622	645	783	791	490
FRESNO, CAL	107	785	460		272	221	154	176	813	301	169	271	347	180	147	241	1005	1046	1145	141
GRANTS PASS, ORE	640	351	173	533	411	754	538	424	280	232	364	804	890	428	467	750	472	623	618	562
KING CITY, CAL	165	821	441	123	319	277	64	145	849	337	216	329	399	153	102	181	1041	1082	1181	225
KLAMATH FALLS, ORE	602	290	282	495	326	716	498	386	342	194	326	766	842	390	429	712	534	551	650	524
LAKEPORT, CAL	380	590	201	273	205	494	233	121	649	137	113	544	620	139	164	447	810	882	981	302
LAKE TAHOE, CAL	379	606	380	272		481	299	211	658	189	119	458	584	215	230	500	850	820	996	301
LEBEC, CAL	39	931	607	146	419	75	263	322	959	447	315	130	201	326	293	110	1151	1192	1297	287
LONG BEACH, CAL	134	1026	702	241	501	20	359	417	1054	542	410	68	104	421	388	114	1246	1268	1386	382
LOS ANGELES, CAL	114	1006	682	221	481		341	397	1034	522	390	58	126	401	368	96	1226	1248	1366	362
MARYSVILLE, CAL	324	570	277	217	103	438	239	128	598	86	48	488	564	132	170	445	790	831	930	246

Figure 11-9 Table of Distances

COBOL has the capability of processing tables of this type, and an explanation is presented on the following pages of the entries required within the COBOL program to process "multi-level" tables. In the example a Rate is to be extracted from a table and printed. The Rate is based upon the City and the Class (Economy or First Class) for air transportation between two Cities. The table is illustrated below.

	(1) San Diego	(2) San Francisco	(3) Santa Barbara	(4) San Jose
ECONOMY (1)	7.50	15.00	12.50	19.00
FIRST CLASS (2)	9.00	17.95	14.50	21.00

Figure 11-10 Multi-Level Table

The chart above illustrates a simulated Passenger Rate Table for air transportation from Los Angeles to the cities listed in the table. For example, if a passenger indicates the need for an airplane ticket to San Francisco, Economy Class, the proper rate can be found in the table as being $15.00

Input for the sample problem is to consist of punched cards containing the Customer Number field, Customer Name field, a Class Code field, and a Location field.

Input

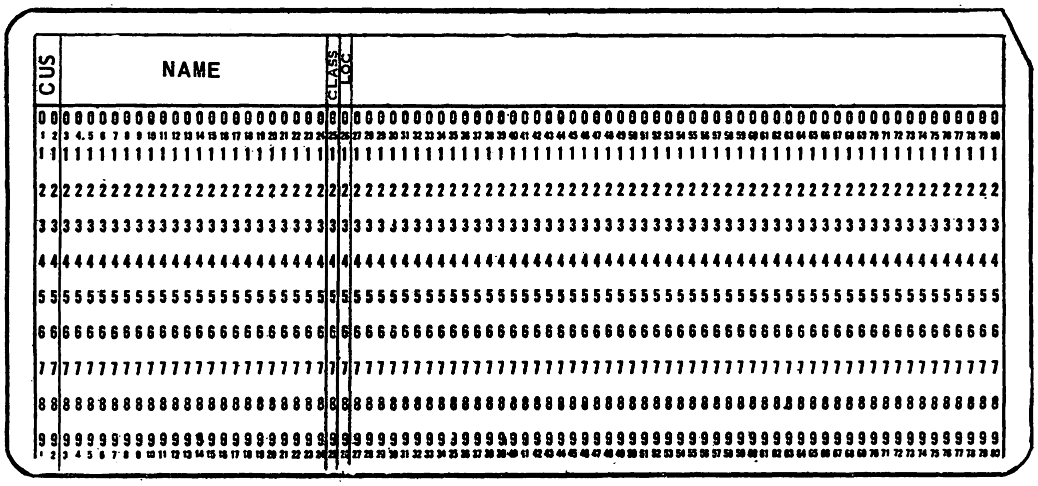


Figure 11-11 Input Data Card

The Class Code Field is based upon the following chart:

PUNCH	MEANING
1	Economy Class
2	First Class

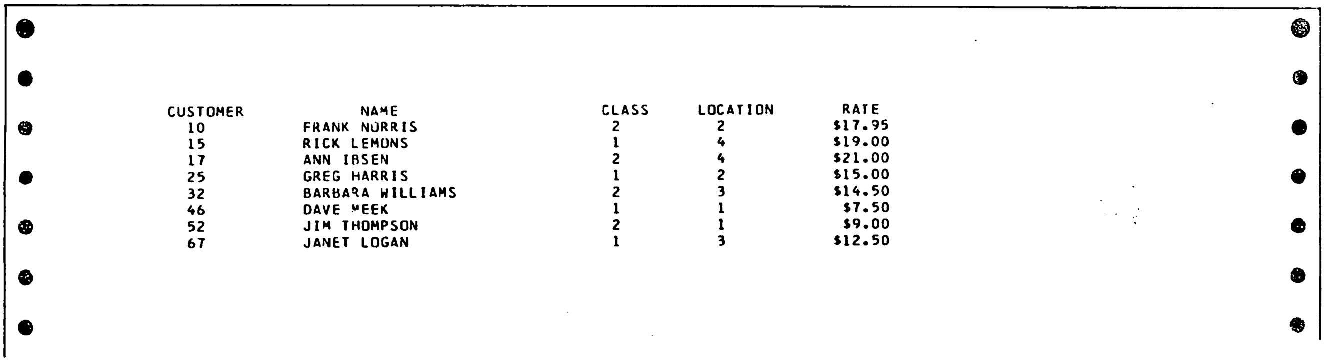
Thus, a "1" punch in the card indicates a request for an Economy Class Flight ticket, a "2" punch in the card indicates a request for a First Class ticket.

The Location Code field is based upon the following chart:

PUNCH	MEANING
1	San Diego
2	San Francisco
3	Santa Barbara
4	San Jose

Output

The output to be produced is a listing of the Customer Number, the Customer Name, each of the Codes, and the Rate.

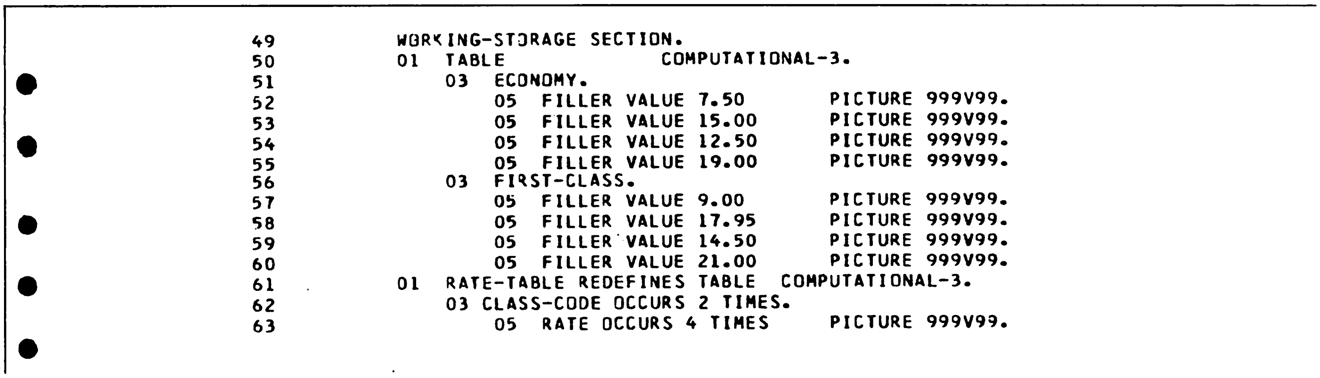


```
CUSTOMER           NAME                  CLASS     LOCATION      RATE
   10      FRANK NORRIS                    2          2         $17.95
   15      RICK LEMONS                     1          4         $19.00
   17      ANN IBSEN                       2          4         $21.00
   25      GREG HARRIS                     1          2         $15.00
   32      BARBARA WILLIAMS                2          3         $14.50
   46      DAVE MEEK                       1          1          $7.50
   52      JIM THOMPSON                    2          1          $9.00
   67      JANET LOGAN                     1          3         $12.50
```

Figure 11-12 Sample Output

WORKING-STORAGE SECTION

The table is set up in WORKING-STORAGE and then redefined so that reference can be made to the table by use of subscripts with the statements in the Procedure Division. The following segment of the COBOL Program for the sample problem illustrates the entries to establish the table.



```
WORKING-STORAGE SECTION.
01  TABLE               COMPUTATIONAL-3.
    03  ECONOMY.
        05  FILLER VALUE 7.50        PICTURE 999V99.
        05  FILLER VALUE 15.00       PICTURE 999V99.
        05  FILLER VALUE 12.50       PICTURE 999V99.
        05  FILLER VALUE 19.00       PICTURE 999V99.
    03  FIRST-CLASS.
        05  FILLER VALUE 9.00        PICTURE 999V99.
        05  FILLER VALUE 17.95       PICTURE 999V99.
        05  FILLER VALUE 14.50       PICTURE 999V99.
        05  FILLER VALUE 21.00       PICTURE 999V99.
01  RATE-TABLE REDEFINES TABLE  COMPUTATIONAL-3.
    03 CLASS-CODE OCCURS 2 TIMES.
        05  RATE OCCURS 4 TIMES      PICTURE 999V99.
```

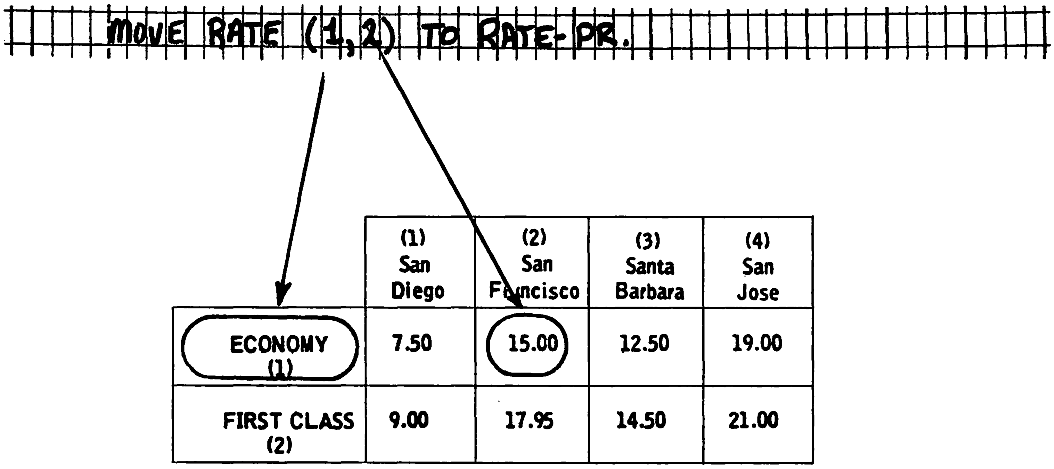
Figure 11-13 WORKING-STORAGE SECTION to describe a Multi-Level Table

In effect, two tables are established, a Table of Rates for Economy Class flights and a Table of Rates for First Class flights. Note how the tables are redefined so that reference may be made to the elements within the table by means of subscripts.

PROCEDURE DIVISION STATEMENTS

Reference to the elements in the table is made by use of double subscripts by the statements in the Procedure Division which are used to reference the table. The following example illustrates the use of a MOVE statement using numeric subscripts.

EXAMPLE 1



```
MOVE RATE (1,2) TO RATE-PR.
```

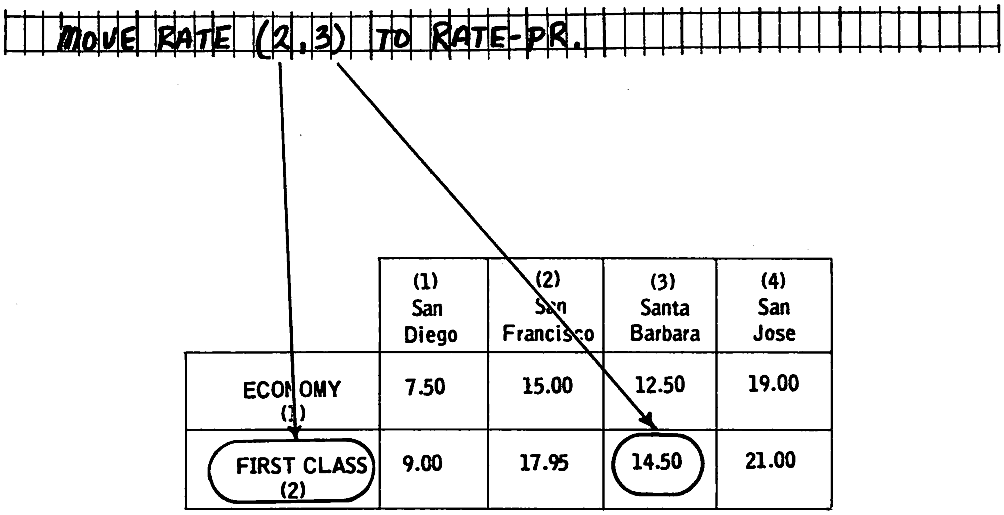
	(1) San Diego	(2) San Francisco	(3) Santa Barbara	(4) San Jose
ECONOMY (1)	7.50	15.00	12.50	19.00
FIRST CLASS (2)	9.00	17.95	14.50	21.00

Figure 11-14 MOVE Statement to reference Table 1, Element 2

In effect the MOVE Statement above directs the computer to "Go to Table 1" (Economy Class Table) and extract the second element from that table.

Example 2 below illustrates the use of the MOVE Statement in which the third element from the "First Class" Table is extracted.

EXAMPLE 2



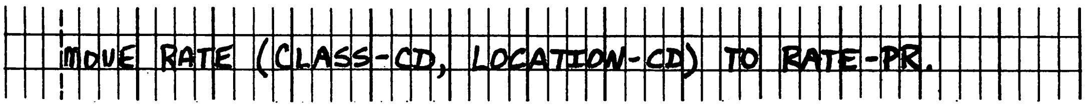
```
MOVE RATE (2,3) TO RATE-PR.
```

	(1) San Diego	(2) San Francisco	(3) Santa Barbara	(4) San Jose
ECONOMY (1)	7.50	15.00	12.50	19.00
FIRST CLASS (2)	9.00	17.95	14.50	21.00

Figure 11-15 MOVE Statement to reference Table 2, Element 3

The following example illustrates the use of data–names within the subscript.

EXAMPLE 3



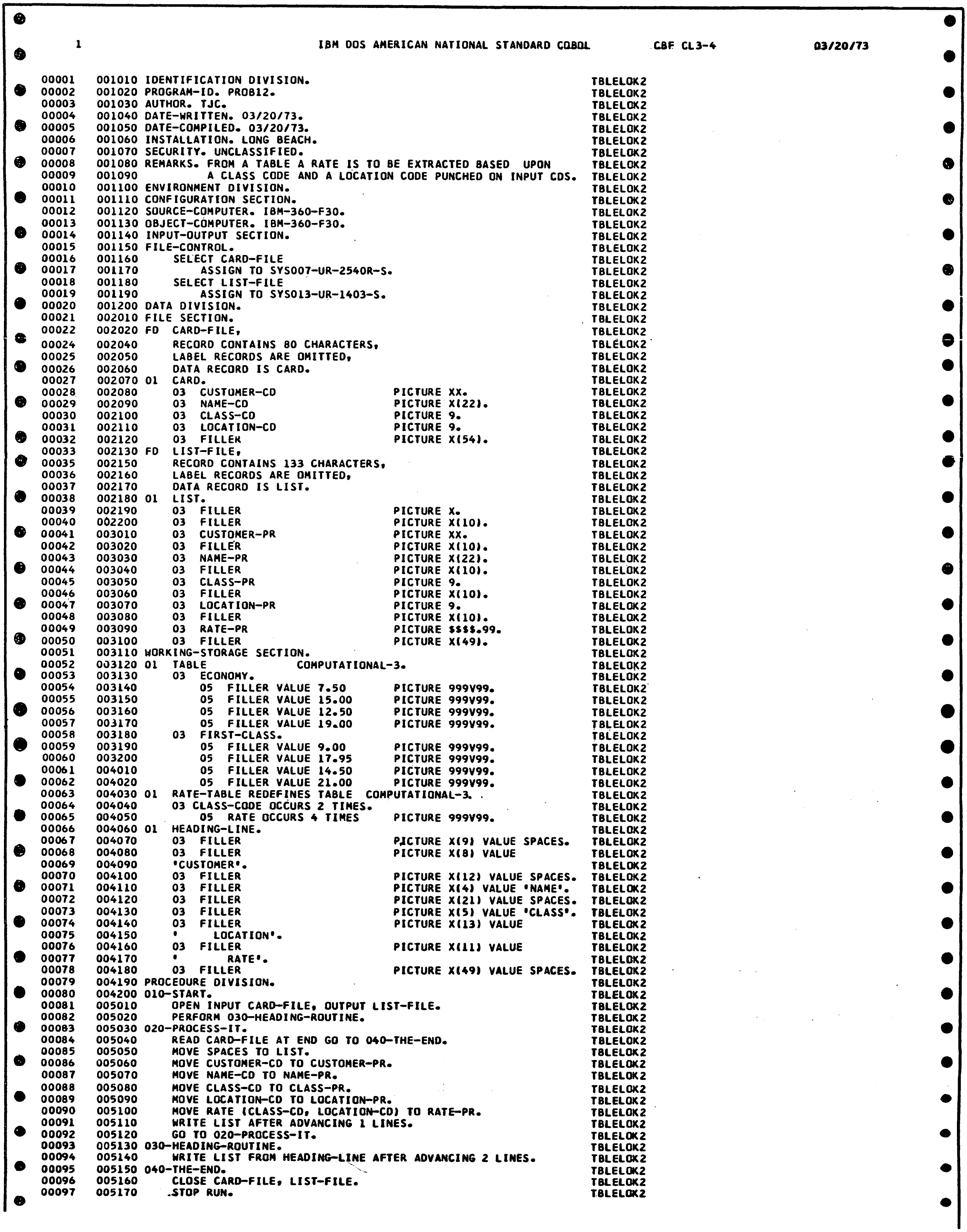
```
MOVE RATE (CLASS-CD, LOCATION-CD) TO RATE-PR.
```

Figure 11-16 COBOL Program

In the example above if CLASS takes on the value of 02 (because 02 is punched in the Class Field of the input data card), and LOCATION takes on the value of 04, from the "First Class" segment of the Table the fourth element is extracted.

The following page contains the complete program to extract a Rate from a multi–level table established within the COBOL program.

COBOL PROGRAM



```
    1                                   IBM DOS AMERICAN NATIONAL STANDARD COBOL         CBF CL3-4          03/20/73

00001   001010 IDENTIFICATION DIVISION.                                         TBLELOK2
00002   001020 PROGRAM-ID. PROB12.                                              TBLELOK2
00003   001030 AUTHOR. TJC.                                                     TBLELOK2
00004   001040 DATE-WRITTEN. 03/20/73.                                          TBLELOK2
00005   001050 DATE-COMPILED. 03/20/73.                                         TBLELOK2
00006   001060 INSTALLATION. LONG BEACH.                                        TBLELOK2
00007   001070 SECURITY. UNCLASSIFIED.                                          TBLELOK2
00008   001080 REMARKS. FROM A TABLE A RATE IS TO BE EXTRACTED BASED  UPON      TBLELOK2
00009   001090          A CLASS CODE AND A LOCATION CODE PUNCHED ON INPUT CDS.  TBLELOK2
00010   001100 ENVIRONMENT DIVISION.                                            TBLELOK2
00011   001110 CONFIGURATION SECTION.                                           TBLELOK2
00012   001120 SOURCE-COMPUTER. IBM-360-F30.                                    TBLELOK2
00013   001130 OBJECT-COMPUTER. IBM-360-F30.                                    TBLELOK2
00014   001140 INPUT-OUTPUT SECTION.                                            TBLELOK2
00015   001150 FILE-CONTROL.                                                    TBLELOK2
00016   001160     SELECT CARD-FILE                                             TBLELOK2
00017   001170         ASSIGN TO SYS007-UR-2540R-S.                             TBLELOK2
00018   001180     SELECT LIST-FILE                                             TBLELOK2
00019   001190         ASSIGN TO SYS013-UR-1403-S.                              TBLELOK2
00020   001200 DATA DIVISION.                                                   TBLELOK2
00021   002010 FILE SECTION.                                                    TBLELOK2
00022   002020 FD  CARD-FILE,                                                   TBLELOK2
00024   002040     RECORD CONTAINS 80 CHARACTERS,                               TBLELOK2
00025   002050     LABEL RECORDS ARE OMITTED,                                   TBLELOK2
00026   002060     DATA RECORD IS CARD.                                         TBLELOK2
00027   002070 01  CARD.                                                        TBLELOK2
00028   002080     03  CUSTOMER-CD                 PICTURE XX.                  TBLELOK2
00029   002090     03  NAME-CD                     PICTURE X(22).               TBLELOK2
00030   002100     03  CLASS-CD                    PICTURE 9.                   TBLELOK2
00031   002110     03  LOCATION-CD                 PICTURE 9.                   TBLELOK2
00032   002120     03  FILLER                      PICTURE X(54).               TBLELOK2
00033   002130 FD  LIST-FILE,                                                   TBLELOK2
00035   002150     RECORD CONTAINS 133 CHARACTERS,                              TBLELOK2
00036   002160     LABEL RECORDS ARE OMITTED,                                   TBLELOK2
00037   002170     DATA RECORD IS LIST.                                         TBLELOK2
00038   002180 01  LIST.                                                        TBLELOK2
00039   002190     03  FILLER                      PICTURE X.                   TBLELOK2
00040   002200     03  FILLER                      PICTURE X(10).               TBLELOK2
00041   003010     03  CUSTOMER-PR                 PICTURE XX.                  TBLELOK2
00042   003020     03  FILLER                      PICTURE X(10).               TBLELOK2
00043   003030     03  NAME-PR                     PICTURE X(22).               TBLELOK2
00044   003040     03  FILLER                      PICTURE X(10).               TBLELOK2
00045   003050     03  CLASS-PR                    PICTURE 9.                   TBLELOK2
00046   003060     03  FILLER                      PICTURE X(10).               TBLELOK2
00047   003070     03  LOCATION-PR                 PICTURE 9.                   TBLELOK2
00048   003080     03  FILLER                      PICTURE X(10).               TBLELOK2
00049   003090     03  RATE-PR                     PICTURE $$$$.99.             TBLELOK2
00050   003100     03  FILLER                      PICTURE X(49).               TBLELOK2
00051   003110 WORKING-STORAGE SECTION.                                         TBLELOK2
00052   003120 01  TABLE             COMPUTATIONAL-3.                           TBLELOK2
00053   003130     03  ECONOMY.                                                 TBLELOK2
00054   003140         05  FILLER VALUE 7.50       PICTURE 999V99.              TBLELOK2
00055   003150         05  FILLER VALUE 15.00      PICTURE 999V99.              TBLELOK2
00056   003160         05  FILLER VALUE 12.50      PICTURE 999V99.              TBLELOK2
00057   003170         05  FILLER VALUE 19.00      PICTURE 999V99.              TBLELOK2
00058   003180     03  FIRST-CLASS.                                             TBLELOK2
00059   003190         05  FILLER VALUE 9.00       PICTURE 999V99.              TBLELOK2
00060   003200         05  FILLER VALUE 17.95      PICTURE 999V99.              TBLELOK2
00061   004010         05  FILLER VALUE 14.50      PICTURE 999V99.              TBLELOK2
00062   004020         05  FILLER VALUE 21.00      PICTURE 999V99.              TBLELOK2
00063   004030 01  RATE-TABLE REDEFINES TABLE  COMPUTATIONAL-3.                 TBLELOK2
00064   004040     03 CLASS-CODE OCCURS 2 TIMES.                                TBLELOK2
00065   004050         05  RATE OCCURS 4 TIMES     PICTURE 999V99.              TBLELOK2
00066   004060 01  HEADING-LINE.                                                TBLELOK2
00067   004070     03  FILLER                      PICTURE X(9) VALUE SPACES.   TBLELOK2
00068   004080     03  FILLER                      PICTURE X(8) VALUE           TBLELOK2
00069   004090     'CUSTOMER'.                                                  TBLELOK2
00070   004100     03  FILLER                      PICTURE X(12) VALUE SPACES.  TBLELOK2
00071   004110     03  FILLER                      PICTURE X(4) VALUE 'NAME'.   TBLELOK2
00072   004120     03  FILLER                      PICTURE X(21) VALUE SPACES.  TBLELOK2
00073   004130     03  FILLER                      PICTURE X(5) VALUE 'CLASS'.  TBLELOK2
00074   004140     03  FILLER                      PICTURE X(13) VALUE          TBLELOK2
00075   004150     '     LOCATION'.                                             TBLELOK2
00076   004160     03  FILLER                      PICTURE X(11) VALUE          TBLELOK2
00077   004170     '       RATE'.                                               TBLELOK2
00078   004180     03  FILLER                      PICTURE X(49) VALUE SPACES.  TBLELOK2
00079   004190 PROCEDURE DIVISION.                                              TBLELOK2
00080   004200 010-START.                                                       TBLELOK2
00081   005010     OPEN INPUT CARD-FILE, OUTPUT LIST-FILE.                      TBLELOK2
00082   005020     PERFORM 030-HEADING-ROUTINE.                                 TBLELOK2
00083   005030 020-PROCESS-IT.                                                  TBLELOK2
00084   005040     READ CARD-FILE AT END GO TO 040-THE-END.                     TBLELOK2
00085   005050     MOVE SPACES TO LIST.                                         TBLELOK2
00086   005060     MOVE CUSTOMER-CD TO CUSTOMER-PR.                             TBLELOK2
00087   005070     MOVE NAME-CD TO NAME-PR.                                     TBLELOK2
00088   005080     MOVE CLASS-CD TO CLASS-PR.                                   TBLELOK2
00089   005090     MOVE LOCATION-CD TO LOCATION-PR.                             TBLELOK2
00090   005100     MOVE RATE (CLASS-CD, LOCATION-CD) TO RATE-PR.                TBLELOK2
00091   005110     WRITE LIST AFTER ADVANCING 1 LINES.                          TBLELOK2
00092   005120     GO TO 020-PROCESS-IT.                                        TBLELOK2
00093   005130 030-HEADING-ROUTINE.                                             TBLELOK2
00094   005140     WRITE LIST FROM HEADING-LINE AFTER ADVANCING 2 LINES.        TBLELOK2
00095   005150 040-THE-END.                                                     TBLELOK2
00096   005160     CLOSE CARD-FILE, LIST-FILE.                                  TBLELOK2
00097   005170     STOP RUN.                                                    TBLELOK2
```

TABLE LOOKUP - EXAMPLE 3

Table Lookup operations provide a valuable tool for the programmer. The previous examples have presented several of the basic programming techniques useful for table lookup operations. In each of those operations, however, a specific element from a table was extracted based upon a control punch in a card. In many applications, there will not necessarily be a direct relationship between the control punch in the card and the elements in the table. In operations of this type it is necessary to "search" the table until a "match" is found corresponding to a control number of the input data card.

The sample problem developed in this example illustrates an operation in which a table is established consisting of an Item Number and a related Item Description. The Input Data Cards contain only the Item Number and Amount. A report is to be prepared listing the Item Number, Item Description, and Amount. To obtain the report the table must be searched for a matching Item Number and, if found, the Item Description extracted from the table.

The following chart illustrates the Item Numbers and related Item Descriptions used in this problem.

ITEM #	DESCRIPTION
008	American Cheese
017	Sweet Butter
024	Asparagus
064	Coffee
069	Dried Peaches
073	Horse Radish
086	Carrots
095	Peas
101	Sweet Potatoes
110	Pineapples
116	Hot Peppers
125	Soda Crackers
137	Rice
138	Tomato Soup
138	Sugar
145	Pears
146	Cocoa
149	Tapioca
152	Nutmeg

Figure 11-17 Table of Item Numbers and related Item Description

WORKING-STORAGE SECTION

Note that the Item Numbers do not begin with "1" and are not consecutive. Thus, after the table is established, it is necessary to compare the Item Number in the card to each of the Item Numbers in the table until a "match" is found, at which time the proper Item Description can be extracted from the table. The following is an illustration of the WORKING-STORAGE Section of a COBOL Program to establish the table.

```
00048   003080 WORKING-STORAGE SECTION.                                         TBLELOK3
00049   003090 77  NUM                          PICTURE S99     VALUE 1          TBLELOK3
00050   003100                                  USAGE IS COMPUTATIONAL.          TBLELOK3
00051   003110 01  TABLE-GROUP.                                                  TBLELOK3
00052   003120     03  ITEM-NUMBER-TABLE.                                        TBLELOK3
00053   003130         05  FILLER               PICTURE XXX     VALUE '008'.     TBLELOK3
00054   003140         05  FILLER               PICTURE XXX     VALUE '017'.     TBLELOK3
00055   003150         05  FILLER               PICTURE XXX     VALUE '024'.     TBLELOK3
00056   003160         05  FILLER               PICTURE XXX     VALUE '064'.     TBLELOK3
00057   003170         05  FILLER               PICTURE XXX     VALUE '069'.     TBLELOK3
00058   003180         05  FILLER               PICTURE XXX     VALUE '073'.     TBLELOK3
00059   003190         05  FILLER               PICTURE XXX     VALUE '086'.     TBLELOK3
00060   003200         05  FILLER               PICTURE XXX     VALUE '095'.     TBLELOK3
00061   004010         05  FILLER               PICTURE XXX     VALUE '101'.     TBLELOK3
00062   004020         05  FILLER               PICTURE XXX     VALUE '110'.     TBLELOK3
00063   004030         05  FILLER               PICTURE XXX     VALUE '116'.     TBLELOK3
00064   004040         05  FILLER               PICTURE XXX     VALUE '125'.     TBLELOK3
00065   004050         05  FILLER               PICTURE XXX     VALUE '137'.     TBLELOK3
00066   004060         05  FILLER               PICTURE XXX     VALUE '138'.     TBLELOK3
00067   004070         05  FILLER               PICTURE XXX     VALUE '139'.     TBLELOK3
00068   004080         05  FILLER               PICTURE XXX     VALUE '145'.     TBLELOK3
00069   004090         05  FILLER               PICTURE XXX     VALUE '146'.     TBLELOK3
00070   004100         05  FILLER               PICTURE XXX     VALUE '149'.     TBLELOK3
00071   004110         05  FILLER               PICTURE XXX     VALUE '152'.     TBLELOK3
00072   004120     03  ITEM-NUMBER REDEFINES ITEM-NUMBER-TABLE                   TBLELOK3
00073   004130         OCCURS 19 TIMES          PICTURE XXX.                     TBLELOK3
00074   004140     03  DESCRIPTION-TABLE.                                        TBLELOK3
00075   004150         05  FILLER       PICTURE X(20) VALUE 'AMERICAN CHEESE'.  TBLELOK3
00076   004160         05  FILLER       PICTURE X(20) VALUE 'SWEET BUTTER'.     TBLELOK3
00077   004170         05  FILLER       PICTURE X(20) VALUE 'ASPARAGUS'.        TBLELOK3
00078   004180         05  FILLER       PICTURE X(20) VALUE 'COFFEE'.           TBLELOK3
00079   004190         05  FILLER       PICTURE X(20) VALUE 'DRIED PEACHES'.    TBLELOK3
00080   004200         05  FILLER       PICTURE X(20) VALUE 'HORSE RADISH'.     TBLELOK3
00081   005010         05  FILLER       PICTURE X(20) VALUE 'CARROTS'.          TBLELOK3
00082   005020         05  FILLER       PICTURE X(20) VALUE 'PEAS'.             TBLELOK3
00083   005030         05  FILLER       PICTURE X(20) VALUE 'SWEET POTATOES'.   TBLELOK3
00084   005040         05  FILLER       PICTURE X(20) VALUE 'PINEAPPLES'.       TBLELOK3
00085   005050         05  FILLER       PICTURE X(20) VALUE 'HOT PEPPERS'.      TBLELOK3
00086   005060         05  FILLER       PICTURE X(20) VALUE 'SODA CRACKERS'.    TBLELOK3
00087   005070         05  FILLER       PICTURE X(20) VALUE 'RICE'.             TBLELOK3
00088   005080         05  FILLER       PICTURE X(20) VALUE 'TOMATO SOUP'.      TBLELOK3
00089   005090         05  FILLER       PICTURE X(20) VALUE 'SUGAR'.            TBLELOK3
00090   005100         05  FILLER       PICTURE X(20) VALUE 'PEARS'.            TBLELOK3
00091   005110         05  FILLER       PICTURE X(20) VALUE 'COCOA'.            TBLELOK3
00092   005120         05  FILLER       PICTURE X(20) VALUE 'TAPIOCA'.          TBLELOK3
00093   005130         05  FILLER       PICTURE X(20) VALUE 'NUTMEG'.           TBLELOK3
00094   005140     03  DESCRIPTION REDEFINES DESCRIPTION-TABLE                   TBLELOK3
00095   005150          OCCURS 19 TIMES PICTURE X(20).                          TBLELOK3
```

Figure 11-18 WORKING-STORAGE SECTION to describe Table of Item Numbers and Table of Item Description

Note that the Level 77 entry NUM is specified as COMPUTATIONAL (binary). NUM will contain the number referenced in the subscripted statements in the Procedure Division. Specifying subscripted numbers as COMPUTATIONAL format (binary) results in greater efficiency in computing the address of the data items.

The effect of the entries in the WORKING-STORAGE Section illustrated in Figure 11-18 is to establish a Table of Item Numbers and a Table of Item Descriptions as indicated below.

008	017	024	064	069	. . . . .

Table Of Item Numbers

AMERICAN CHEESE	SWEET BUTTER	ASPARAGUS	COFFEE . . . .

Table Of Descriptions

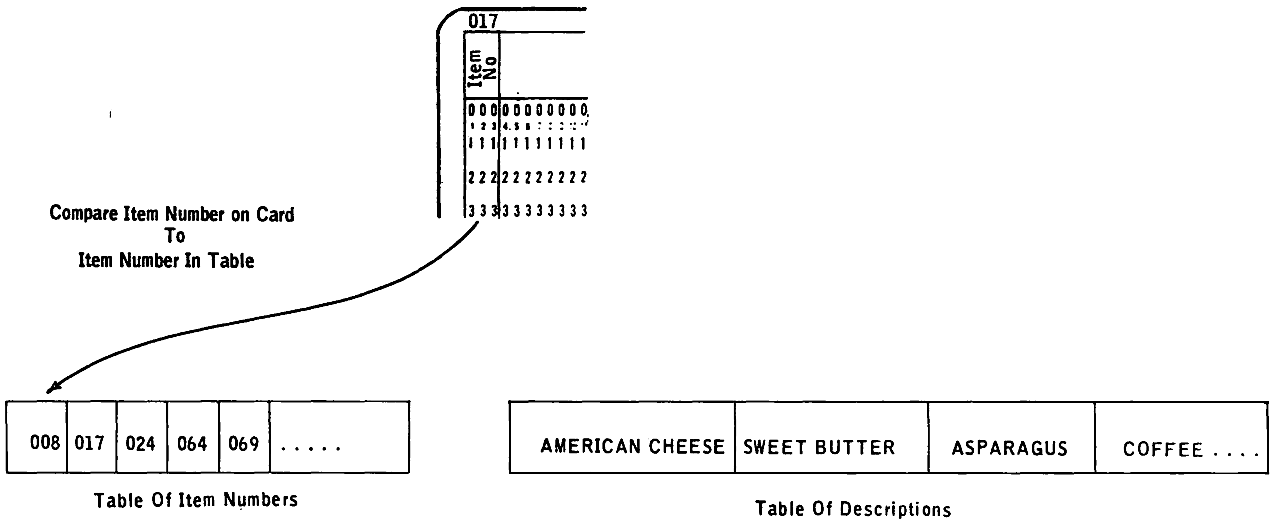
Figure 11-19 Schematic of Table of Item Numbers and Table of Item Description

The basic logic in searching a table is to compare the Item Number on the input data card to the Item Number contained in the first element of the Table of Item Numbers. If the Item Numbers are equal, the related element from the Table of Descriptions is extracted. If the Item Number on the card is not equal to the Item Number in the first element of the Table of Item Numbers, the data-name used as the subscript is incremented by "1". When, the sequence of instructions is repeated, the second element in the Table of Item Numbers is compared to the Item Number punched on the card. This process continues until a match is found.

SEARCHING A TABLE

The following diagrams illustrate the basic steps in searching a table to extract the proper Item Description.

Step 1: **The Item Number on the Card is compared to the Item Number contained in the first element of the table. If they are unequal the subscript used in the statements in the Procedure Division is incremented by "1". When the sequence of instructions is repeated, the Item Number in the second element of the table will be compared with the Item Number on the input data card.**


017
Item No
Compare Item Number on Card To Item Number In Table
008
017
024
064
069
.....
Table Of Item Numbers
AMERICAN CHEESE
SWEET BUTTER
ASPARAGUS
COFFEE ....
Table Of Descriptions

Step 2: **When an equal condition occurs the related Item Description is extracted from the Item Description Table.**

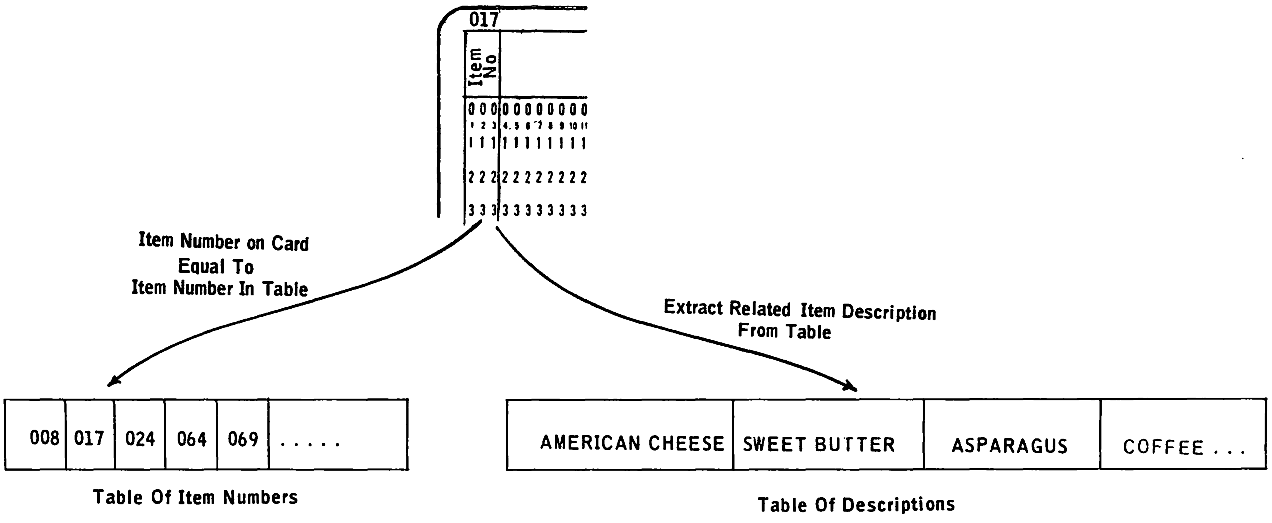

017
Item No
Item Number on Card Equal To Item Number In Table
Extract Related Item Description From Table
008
017
024
064
069
.....
Table Of Item Numbers
AMERICAN CHEESE
SWEET BUTTER
ASPARAGUS
COFFEE ...
Table Of Descriptions

Figure 11-20 Steps in searching a table

The following segment of the Procedure Division illustrates the entries to perform the table lookup operation previously explained.

```
006160 020-NOTE.                                                            TBLELOK3
006170     NOTE ****                                                        TBLELOK3
006180     *   THIS ROUTINE IS ENTERED TO PERFORM THE TABLE SEARCH          TBLELOK3
006190     *   OPERATION. THE ITEM NUMBER IN THE INPUT CARD IS              TBLELOK3
006200     *   COMPARED TO EACH ITEM NUMBER IN THE TABLE. WHEN AN           TBLELOK3
007010     *   EQUAL ITEM NUMBER IS FOUND, THE CORRESPONDING                TBLELOK3
007020     *   DESCRIPTION FROM THE DESCRIPTION TABLE IS MOVED TO           TBLELOK3
007030     *   THE PRINTER OUTPUT AREA.                                     TBLELOK3
007040     *********.                                                       TBLELOK3
007050                                                                      TBLELOK3
007060 020-SEARCH.                                                          TBLELOK3
007070     IF ITEM-NUMBER (NUM) = ITEM-CD-X                                 TBLELOK3
007080         GO TO 030-FOUND.                                             TBLELOK3
007090     IF NUM = 19                                                      TBLELOK3
007100         MOVE '***ITEM NOT FOUND***' TO DESCRIPTION-PL                TBLELOK3
007105         GO TO 040-PRINT                                              TBLELOK3
007110     ELSE ADD 1 TO NUM                                                TBLELOK3
007120         GO TO 020-SEARCH.                                            TBLELOK3
007130                                                                      TBLELOK3
007140 030-NOTE.                                                            TBLELOK3
007150     NOTE ****                                                        TBLELOK3
007160     *   THIS ROUTINE IS ENTERED WHEN THE PROPER ITEM                 TBLELOK3
007170     *   NUMBER HAS BEEN FOUND.                                       TBLELOK3
007180     *********.                                                       TBLELOK3
007190                                                                      TBLELOK3
007200 030-FOUND.                                                           TBLELOK3
008010     MOVE DESCRIPTION (NUM) TO DESCRIPTION-PL.                        TBLELOK3
008020     MOVE AMOUNT-CD TO AMOUNT-PL.                                     TBLELOK3
```

Figure 11-21 Segment of Procedure Division to search a Table

In the 020-SEARCH Routine, which is used to locate the proper item number in the ITEM-NUMBER-TABLE, it can be seen that the first IF statement compares the item number in the card (ITEM-CD-X) with an Item Number in the table. If an equal item number is found, the 030-FOUND routine is entered to extract the corresponding description from the DESCRIPTION-TABLE.

If an equal item number is not found, the value in the subscript NUM is checked to determine if all of the entries in the item table have been examined, that is, if the value NUM is 19, then all nineteen of the entries in the item table have been compared to the item number in the card and an equal condition has not been found. When this occurs, it indicates that the item number in the card is in error. Therefore, an error message is moved to the output area instead of the description so that the card in error can be identified.

If all of the entries have not been examined, that is, if the value in NUM is not equal to 19, then the value in NUM is incremented by 1 and the 020-SEARCH Routine is again executed to search for the proper item number.

There are numerous variations and applications of table lookup. This chapter has introduced some of the basic techniques.

The following pages illustrate the complete documentation for the Table Lookup application in which a table is to be searched for an Item Number so that a related Item Description can be extracted from the table.

SAMPLE PROBLEM

Input

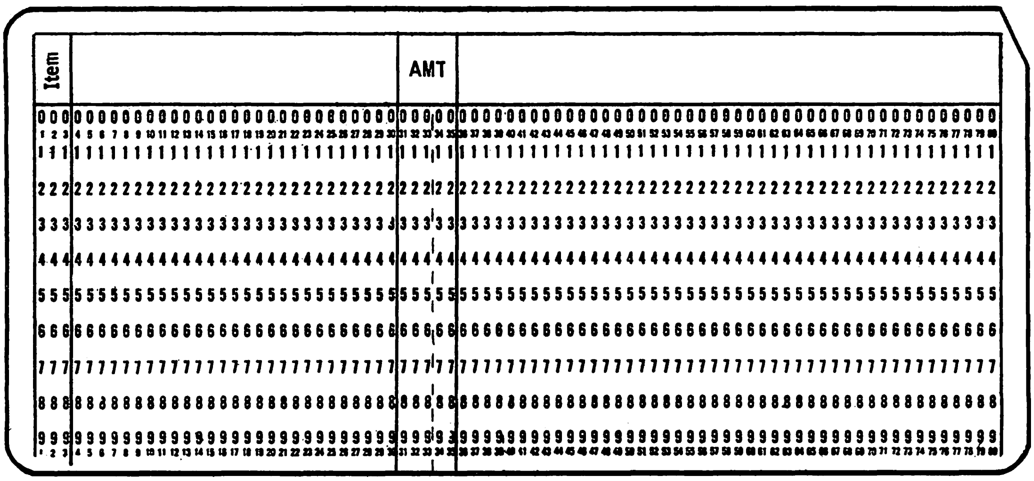
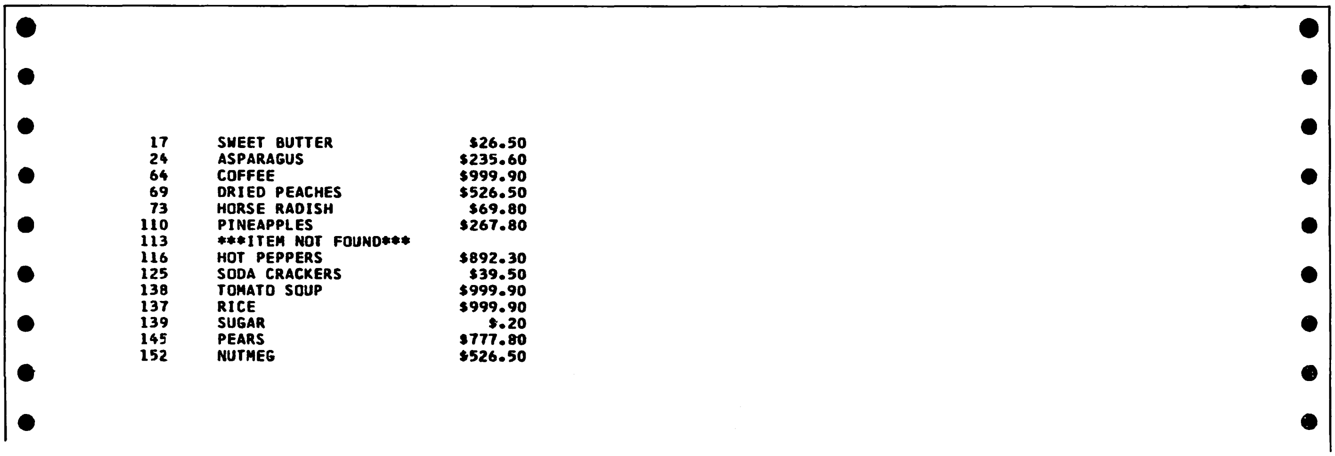


Figure 11-22 Input

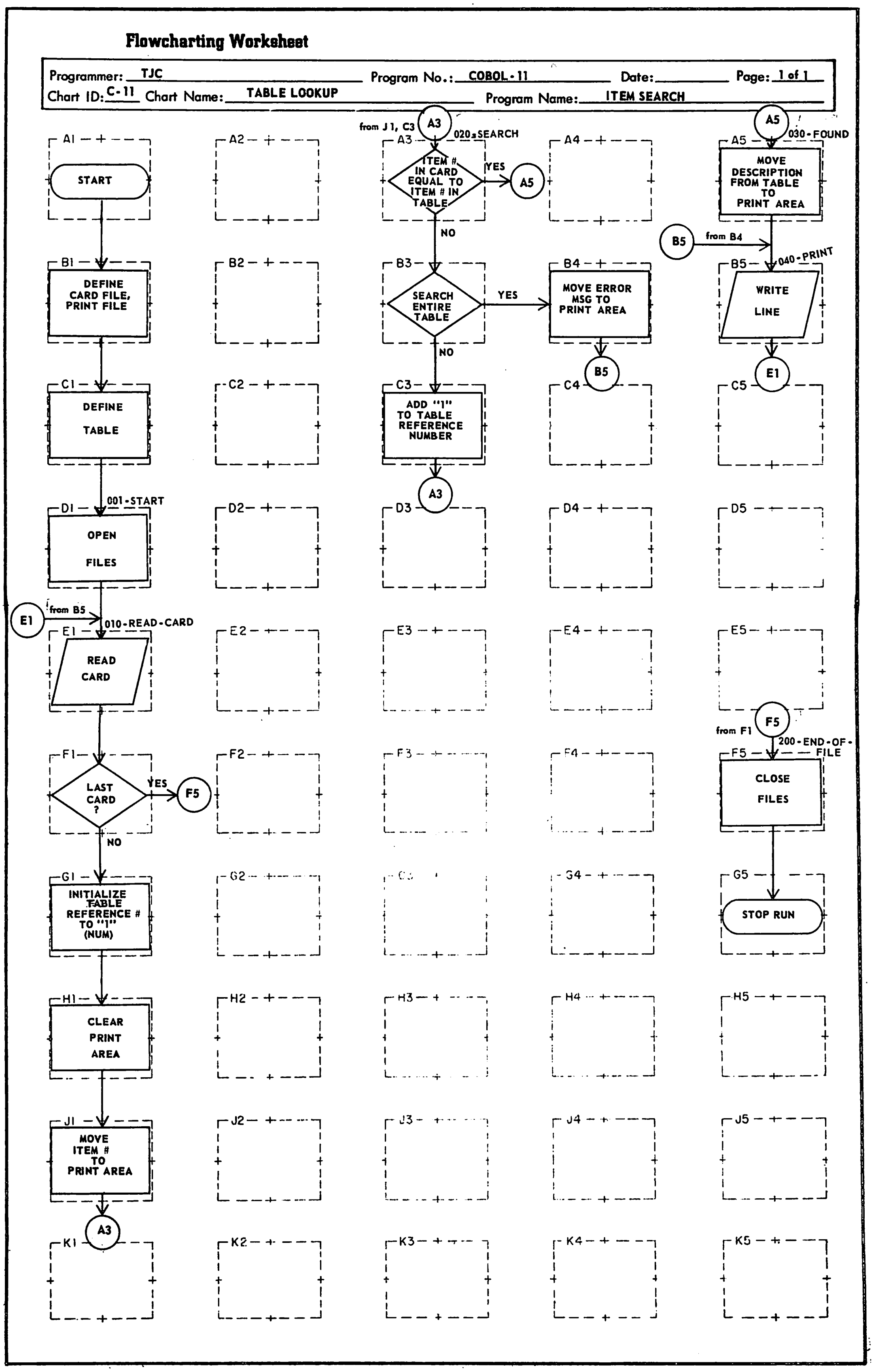
Output



```
 17     SWEET BUTTER                $26.50
 24     ASPARAGUS                  $235.60
 64     COFFEE                     $999.90
 69     DRIED PEACHES              $526.50
 73     HORSE RADISH                $69.80
110     PINEAPPLES                 $267.80
113     ***ITEM NOT FOUND***
116     HOT PEPPERS                $892.30
125     SODA CRACKERS               $39.50
138     TOMATO SOUP                $999.90
137     RICE                       $999.90
139     SUGAR                         $.20
145     PEARS                      $777.80
152     NUTMEG                     $526.50
```

Figure 11-23 Output

Flowchart



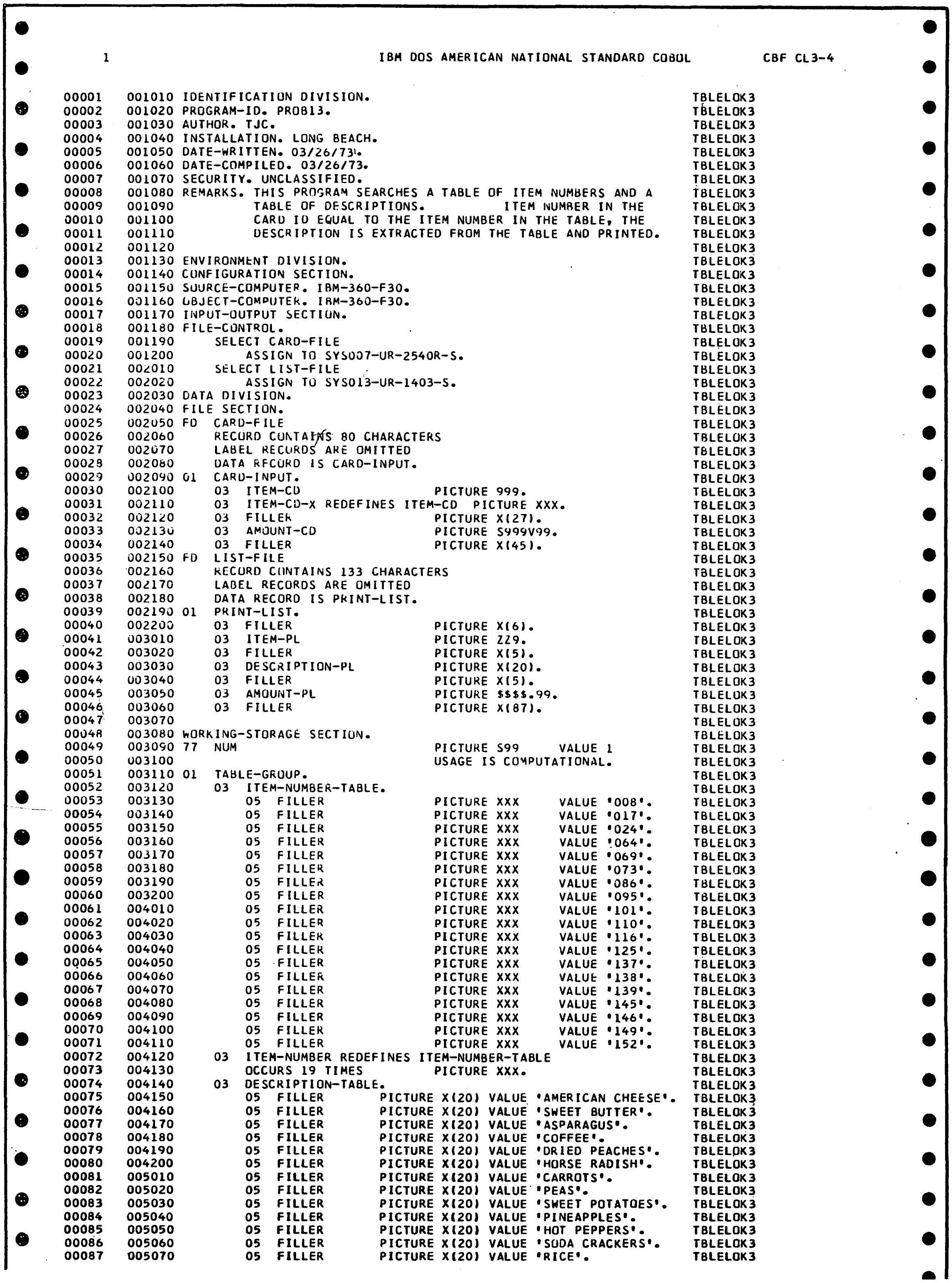
Flowcharting Worksheet

Programmer: TJC Program No.: COBOL-11 Date: Page: 1 of 1
Chart ID: C-11 Chart Name: TABLE LOOKUP Program Name: ITEM SEARCH

A1
START
B1
DEFINE CARD FILE, PRINT FILE
C1
DEFINE TABLE
001-START
D1
OPEN FILES
E1
from B5
010-READ-CARD
E1
READ CARD
F1
LAST CARD ?
YES
F5
NO
G1
INITIALIZE TABLE REFERENCE # TO "1" (NUM)
H1
CLEAR PRINT AREA
J1
MOVE ITEM # TO PRINT AREA
A3
K1
A2
B2
C2
D2
E2
F2
G2
H2
J2
K2
from J1, C3
A3
020-SEARCH
A3
ITEM # IN CARD EQUAL TO ITEM # IN TABLE
YES
A5
NO
B3
SEARCH ENTIRE TABLE
YES
NO
C3
ADD "1" TO TABLE REFERENCE NUMBER
A3
D3
E3
F3
G3
H3
J3
K3
A4
B4
MOVE ERROR MSG TO PRINT AREA
B5
C4
D4
E4
F4
G4
H4
J4
K4
A5
030-FOUND
A5
MOVE DESCRIPTION FROM TABLE TO PRINT AREA
B5
from B4
040-PRINT
B5
WRITE LINE
E1
C5
D5
E5
F5
from F1
200-END-OF-FILE
F5
CLOSE FILES
G5
STOP RUN
H5
J5
K5

Figure 11-24 Flowchart

COBOL Program



```
     1                                  IBM DOS AMERICAN NATIONAL STANDARD COBOL         CBF CL3-4

00001   001010 IDENTIFICATION DIVISION.                                         TBLELOK3
00002   001020 PROGRAM-ID. PROB13.                                              TBLELOK3
00003   001030 AUTHOR. TJC.                                                     TBLELOK3
00004   001040 INSTALLATION. LONG BEACH.                                        TBLELOK3
00005   001050 DATE-WRITTEN. 03/26/73.                                          TBLELOK3
00006   001060 DATE-COMPILED. 03/26/73.                                         TBLELOK3
00007   001070 SECURITY. UNCLASSIFIED.                                          TBLELOK3
00008   001080 REMARKS. THIS PROGRAM SEARCHES A TABLE OF ITEM NUMBERS AND A     TBLELOK3
00009   001090          TABLE OF DESCRIPTIONS.          ITEM NUMBER IN THE      TBLELOK3
00010   001100          CARD ID EQUAL TO THE ITEM NUMBER IN THE TABLE, THE      TBLELOK3
00011   001110          DESCRIPTION IS EXTRACTED FROM THE TABLE AND PRINTED.    TBLELOK3
00012   001120                                                                  TBLELOK3
00013   001130 ENVIRONMENT DIVISION.                                            TBLELOK3
00014   001140 CONFIGURATION SECTION.                                           TBLELOK3
00015   001150 SOURCE-COMPUTER. IBM-360-F30.                                    TBLELOK3
00016   001160 OBJECT-COMPUTER. IBM-360-F30.                                    TBLELOK3
00017   001170 INPUT-OUTPUT SECTION.                                            TBLELOK3
00018   001180 FILE-CONTROL.                                                    TBLELOK3
00019   001190     SELECT CARD-FILE                                             TBLELOK3
00020   001200         ASSIGN TO SYS007-UR-2540R-S.                             TBLELOK3
00021   002010     SELECT LIST-FILE                                             TBLELOK3
00022   002020         ASSIGN TO SYS013-UR-1403-S.                              TBLELOK3
00023   002030 DATA DIVISION.                                                   TBLELOK3
00024   002040 FILE SECTION.                                                    TBLELOK3
00025   002050 FD  CARD-FILE                                                    TBLELOK3
00026   002060     RECORD CONTAINS 80 CHARACTERS                                TBLELOK3
00027   002070     LABEL RECORDS ARE OMITTED                                    TBLELOK3
00028   002080     DATA RECORD IS CARD-INPUT.                                   TBLELOK3
00029   002090 01  CARD-INPUT.                                                  TBLELOK3
00030   002100     03  ITEM-CD                 PICTURE 999.                     TBLELOK3
00031   002110     03  ITEM-CD-X REDEFINES ITEM-CD  PICTURE XXX.                TBLELOK3
00032   002120     03  FILLER                  PICTURE X(27).                   TBLELOK3
00033   002130     03  AMOUNT-CD               PICTURE S999V99.                 TBLELOK3
00034   002140     03  FILLER                  PICTURE X(45).                   TBLELOK3
00035   002150 FD  LIST-FILE                                                    TBLELOK3
00036   002160     RECORD CONTAINS 133 CHARACTERS                               TBLELOK3
00037   002170     LABEL RECORDS ARE OMITTED                                    TBLELOK3
00038   002180     DATA RECORD IS PRINT-LIST.                                   TBLELOK3
00039   002190 01  PRINT-LIST.                                                  TBLELOK3
00040   002200     03  FILLER                  PICTURE X(6).                    TBLELOK3
00041   003010     03  ITEM-PL                 PICTURE ZZ9.                     TBLELOK3
00042   003020     03  FILLER                  PICTURE X(5).                    TBLELOK3
00043   003030     03  DESCRIPTION-PL          PICTURE X(20).                   TBLELOK3
00044   003040     03  FILLER                  PICTURE X(5).                    TBLELOK3
00045   003050     03  AMOUNT-PL               PICTURE $$$$.99.                 TBLELOK3
00046   003060     03  FILLER                  PICTURE X(87).                   TBLELOK3
00047   003070                                                                  TBLELOK3
00048   003080 WORKING-STORAGE SECTION.                                         TBLELOK3
00049   003090 77  NUM                         PICTURE S99     VALUE 1          TBLELOK3
00050   003100                                 USAGE IS COMPUTATIONAL.          TBLELOK3
00051   003110 01  TABLE-GROUP.                                                 TBLELOK3
00052   003120     03  ITEM-NUMBER-TABLE.                                       TBLELOK3
00053   003130         05  FILLER              PICTURE XXX     VALUE '008'.     TBLELOK3
00054   003140         05  FILLER              PICTURE XXX     VALUE '017'.     TBLELOK3
00055   003150         05  FILLER              PICTURE XXX     VALUE '024'.     TBLELOK3
00056   003160         05  FILLER              PICTURE XXX     VALUE '064'.     TBLELOK3
00057   003170         05  FILLER              PICTURE XXX     VALUE '069'.     TBLELOK3
00058   003180         05  FILLER              PICTURE XXX     VALUE '073'.     TBLELOK3
00059   003190         05  FILLER              PICTURE XXX     VALUE '086'.     TBLELOK3
00060   003200         05  FILLER              PICTURE XXX     VALUE '095'.     TBLELOK3
00061   004010         05  FILLER              PICTURE XXX     VALUE '101'.     TBLELOK3
00062   004020         05  FILLER              PICTURE XXX     VALUE '110'.     TBLELOK3
00063   004030         05  FILLER              PICTURE XXX     VALUE '116'.     TBLELOK3
00064   004040         05  FILLER              PICTURE XXX     VALUE '125'.     TBLELOK3
00065   004050         05  FILLER              PICTURE XXX     VALUE '137'.     TBLELOK3
00066   004060         05  FILLER              PICTURE XXX     VALUE '138'.     TBLELOK3
00067   004070         05  FILLER              PICTURE XXX     VALUE '139'.     TBLELOK3
00068   004080         05  FILLER              PICTURE XXX     VALUE '145'.     TBLELOK3
00069   004090         05  FILLER              PICTURE XXX     VALUE '146'.     TBLELOK3
00070   004100         05  FILLER              PICTURE XXX     VALUE '149'.     TBLELOK3
00071   004110         05  FILLER              PICTURE XXX     VALUE '152'.     TBLELOK3
00072   004120     03  ITEM-NUMBER REDEFINES ITEM-NUMBER-TABLE                  TBLELOK3
00073   004130         OCCURS 19 TIMES         PICTURE XXX.                     TBLELOK3
00074   004140     03  DESCRIPTION-TABLE.                                       TBLELOK3
00075   004150         05  FILLER       PICTURE X(20) VALUE 'AMERICAN CHEESE'.  TBLELOK3
00076   004160         05  FILLER       PICTURE X(20) VALUE 'SWEET BUTTER'.     TBLELOK3
00077   004170         05  FILLER       PICTURE X(20) VALUE 'ASPARAGUS'.        TBLELOK3
00078   004180         05  FILLER       PICTURE X(20) VALUE 'COFFEE'.           TBLELOK3
00079   004190         05  FILLER       PICTURE X(20) VALUE 'DRIED PEACHES'.    TBLELOK3
00080   004200         05  FILLER       PICTURE X(20) VALUE 'HORSE RADISH'.     TBLELOK3
00081   005010         05  FILLER       PICTURE X(20) VALUE 'CARROTS'.          TBLELOK3
00082   005020         05  FILLER       PICTURE X(20) VALUE 'PEAS'.             TBLELOK3
00083   005030         05  FILLER       PICTURE X(20) VALUE 'SWEET POTATOES'.   TBLELOK3
00084   005040         05  FILLER       PICTURE X(20) VALUE 'PINEAPPLES'.       TBLELOK3
00085   005050         05  FILLER       PICTURE X(20) VALUE 'HOT PEPPERS'.      TBLELOK3
00086   005060         05  FILLER       PICTURE X(20) VALUE 'SODA CRACKERS'.    TBLELOK3
00087   005070         05  FILLER       PICTURE X(20) VALUE 'RICE'.             TBLELOK3
```

Figure 11-25 COBOL Program

```
005080         05  FILLER       PICTURE X(20) VALUE 'TOMATO SOUP'.      TBLELOK3
005090         05  FILLER       PICTURE X(20) VALUE 'SUGAR'.            TBLELOK3
005100         05  FILLER       PICTURE X(20) VALUE 'PEARS'.            TBLELOK3
005110         05  FILLER       PICTURE X(20) VALUE 'COCOA'.            TBLELOK3
005120         05  FILLER       PICTURE X(20) VALUE 'TAPIOCA'.          TBLELOK3
005130         05  FILLER       PICTURE X(20) VALUE 'NUTMEG'.           TBLELOK3
005140     03  DESCRIPTION REDEFINES DESCRIPTION-TABLE                  TBLELOK3
005150          OCCURS 19 TIMES PICTURE X(20).                          TBLELOK3
005160 PROCEDURE DIVISION.                                              TBLELOK3
005170 001-NOTE.                                                        TBLELOK3
005180     NOTE ***                                                     TBLELOK3
005190     *   THIS PROGRAM READS AN ITEM CARD AND, USING THE ITEM      TBLELOK3
005200     *   NUMBER SEARCHES THE ITEM-NUMBER-TABLE UNTIL AN EQUAL     TBLELOK3
006010     *   ITEM NUMBER IS FOUND. IT THEN EXTRACTS THE CORRESPONDING TBLELOK3
006020     *   DESCRIPTION FROM THE DESCRIPTION-TABLE AND THE REPORT    TBLELOK3
006030     *   IS PRINTED. IF A CORRESPONDING ITEM NUMBER IS NOT        TBLELOK3
006040     *   FOUND IN THE TABLE, AN ERROR MESSAGE IS PRINTED.         TBLELOK3
006050     ********.                                                    TBLELOK3
006060                                                                  TBLELOK3
006070 001-START.                                                       TBLELOK3
006080     OPEN INPUT CARD-FILE, OUTPUT LIST-FILE.                      TBLELOK3
006090                                                                  TBLELOK3
006100 010-READ-CARD.                                                   TBLELOK3
006110     READ CARD-FILE AT END GO TO 200-END-OF-FILE.                 TBLELOK3
006120     MOVE 1 TO NUM.                                               TBLELOK3
006130     MOVE SPACES TO PRINT-LIST.                                   TBLELOK3
006140     MOVE ITEM-CD TO ITEM-PL.                                     TBLELOK3
006150                                                                  TBLELOK3
006160 020-NOTE.                                                        TBLELOK3
006170     NOTE ****                                                    TBLELOK3
006180     *   THIS ROUTINE IS ENTERED TO PERFORM THE TABLE SEARCH      TBLELOK3
006190     *   OPERATION. THE ITEM NUMBER IN THE INPUT CARD IS          TBLELOK3
006200     *   COMPARED TO EACH ITEM NUMBER IN THE TABLE. WHEN AN       TBLELOK3
007010     *   EQUAL ITEM NUMBER IS FOUND, THE CORRESPONDING            TBLELOK3
007020     *   DESCRIPTION FROM THE DESCRIPTION TABLE IS MOVED TO       TBLELOK3
007030     *   THE PRINTER OUTPUT AREA.                                 TBLELOK3
007040     *********.                                                   TBLELOK3
007050                                                                  TBLELOK3
007060 020-SEARCH.                                                      TBLELOK3
007070     IF ITEM-NUMBER (NUM) = ITEM-CD-X                             TBLELOK3
007080         GO TO 030-FOUND.                                         TBLELOK3
007090     IF NUM = 19                                                  TBLELOK3
007100         MOVE '***ITEM NOT FOUND***' TO DESCRIPTION-PL            TBLELOK3
007105         GO TO 040-PRINT                                          TBLELOK3
007110     ELSE ADD 1 TO NUM                                            TBLELOK3
007120         GO TO 020-SEARCH.                                        TBLELOK3
007130                                                                  TBLELOK3
007140 030-NOTE.                                                        TBLELOK3
007150     NOTE ****                                                    TBLELOK3
007160     *   THIS ROUTINE IS ENTERED WHEN THE PROPER ITEM             TBLELOK3
007170     *   NUMBER HAS BEEN FOUND.                                   TBLELOK3
007180     *********.                                                   TBLELOK3
007190                                                                  TBLELOK3
007200 030-FOUND.                                                       TBLELOK3
008010     MOVE DESCRIPTION (NUM) TO DESCRIPTION-PL.                    TBLELOK3
008020     MOVE AMOUNT-CD TO AMOUNT-PL.                                 TBLELOK3
008030                                                                  TBLELOK3
008040 040-PRINT.                                                       TBLELOK3
008050     WRITE PRINT-LIST AFTER ADVANCING 1 LINES.                    TBLELOK3
008060     GO TO 010-READ-CARD.                                         TBLELOK3
008070                                                                  TBLELOK3
008080 200-NOTE.                                                        TBLELOK3
008090     NOTE ****                                                    TBLELOK3
008100     *   THIS ROUTINE IS ENTERED AFTER ALL OF THE ITEM CARDS      TBLELOK3
008110     *   HAVE BEEN PROCESSED.                                     TBLELOK3
008120     *********.                                                   TBLELOK3
008130                                                                  TBLELOK3
008140 200-END-OF-FILE.                                                 TBLELOK3
008150     CLOSE CARD-FILE, LIST-FILE.                                  TBLELOK3
008160     STOP RUN.                                                    TBLELOK3
```

CHAPTER 11

PROGRAMMING ASSIGNMENT 1

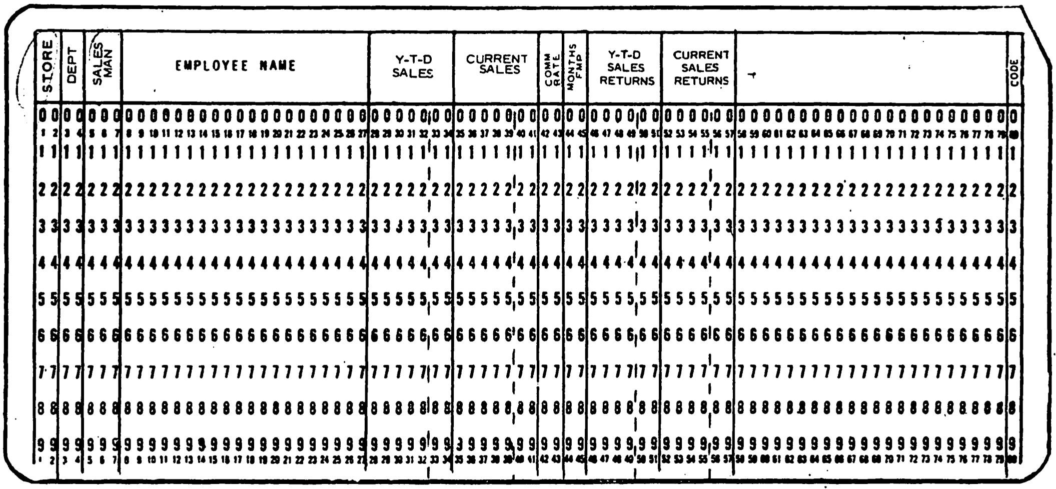
TABLE LOOKUP

INSTRUCTIONS

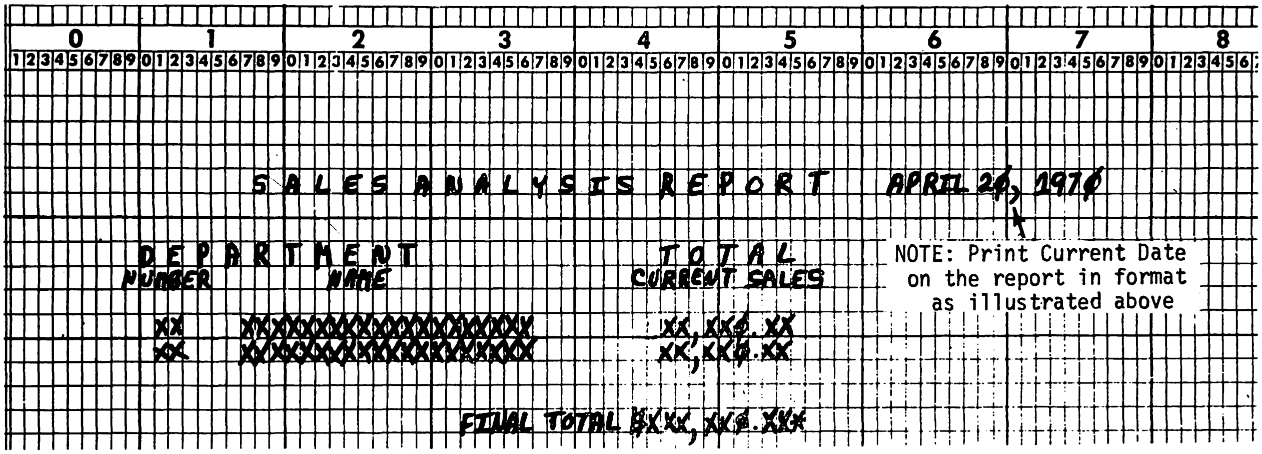
From the Sales cards illustrated below a Departmental Sales Analysis Report is to be produced. This is to be a group printed report listing total departmental Current Sales. The department name is to be printed on a report. The Department Numbers and related names are given below.

DEPT. NO.	NAME
01	AUTOMOTIVE
02	CLOTHING
03	FLOOR COVERING
04	FURNITURE
05	GARDEN SUPPLIES
06	HARDWARE
07	HOUSEHOLD APPLIANCES
08	JEWELRY
09	SPORTING GOODS

INPUT: Sales Cards – Test data is in the appendices.


STORE
DEPT
SALES MAN
EMPLOYEE NAME
Y-T-D SALES
CURRENT SALES
COMM RATE
MONTHS EMP
Y-T-D SALES RETURNS
CURRENT SALES RETURNS
CODE

OUTPUT: Department Sales Analysis Report



```
                      SALES ANALYSIS REPORT            APRIL 20, 1970

          DEPARTMENT                            TOTAL
        NUMBER      NAME                    CURRENT SALES

          XX    XXXXXXXXXXXXXXXXXXXX          XX,XX0.XX
          XX    XXXXXXXXXXXXXXXXXXXX          XX,XX0.XX

                             FINAL TOTAL $XXX,XX0.XX
```

NOTE: Print Current Date on the report in format as illustrated above

CHAPTER 11

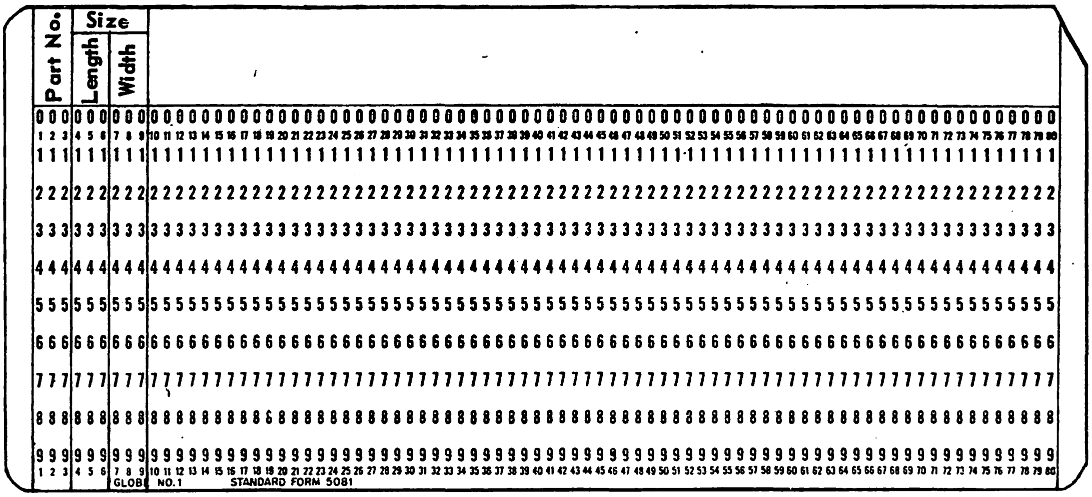
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

Write the COBOL program to create a Shipping Report using the tables illustrated on page 11.24.

INPUT: Sales Cards

Input is to consist of Sales cards that contain the Part Number (which indicates a type of metal), and the size (length and width) of products ordered. The input cards are illustrated below.


Part No.
Size
Length
Width
GLOBE NO. 1 STANDARD FORM 5081

The following charts are used to determine the metal cost, total weight, and shipping charges.

CHART I

WEIGHT/COST PER SQUARE FOOT

PART NUMBER	DESCRIPTION	WEIGHT SQ/FT	COST SQ/FT
100	¼ ALUMINUM ALLOY	1 lb.	1.25
101	½ ALUMINUM ALLOY	2 lb.	2.00
102	¾ ALUMINUM ALLOY	3 lb.	2.80
120	¼ MAGNESIUM	1 lb.	3.75
121	½ MAGNESIUM	2 lb.	7.00
122	¾ MAGNESIUM	3 lb.	11.00

CHART II

SHIPPING CHARGES
AREA 1

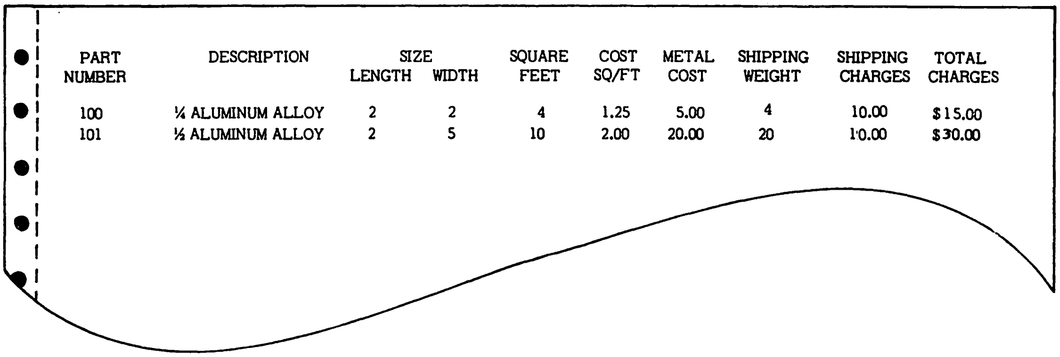
POUNDS	SHIPPING CHARGES
1-50	10.00
51-100	20.00
101-200	45.00
201-300	85.00
301-400	120.00
401-500	150.00

NOTE: MAXIMUM POUNDAGE 500 LBS.

OUTPUT: Shipping Report

The output that is to be obtained is illustrated below. Note that the Metal Cost is obtained by : (1) determining the number of square feet of the metal type desired by multiplying the length field by the width field on the input data cards; (2) determining the Cost Per Square Foot from Chart I; (3) multiplying the number of square feet by the Cost Per Square Foot. For example in the report below Part Number 100 is 2 feet in length and 2 feet in width for a total of 4 square feet. From Chart I it is determined that the Cost Per Square Foot is $1.25. By multiplying the number of square feet (4) by the Cost Per Square Foot ($1.25), the Metal Cost is obtained.

In addition the Shipping Weight and the Shipping Charges are to be determined from Charts I and II.



```
PART        DESCRIPTION           SIZE        SQUARE  COST   METAL   SHIPPING  SHIPPING  TOTAL
NUMBER                       LENGTH   WIDTH   FEET    SQ/FT  COST    WEIGHT    CHARGES   CHARGES

 100    ¼ ALUMINUM ALLOY       2        2       4     1.25    5.00      4       10.00    $15.00
 101    ½ ALUMINUM ALLOY       2        5      10     2.00   20.00     20       10.00    $30.00
```

TEST DATA – ASSIGNMENT NO. 2

Part No. (col 1-3)	Length (col 4-6)	Width (col 7-9)
100	002	002
101	002	005
120	008	020
121	010	020
122	003	050
122	010	050

CHAPTER 12

SORT VERB

SET/SEARCH TABLE LOOKUP

INTRODUCTION

In the previous chapter concerning table lookup, the information which was stored in the tables was defined through the use of the VALUE clause and the REDEFINES statements in the Working-Storage Section of the COBOL program. In many applications, the data which must be stored within tables cannot be defined within the program either because the data is rapidly changing or because there is a variable amount of data which will be processed with each run of the program. Therefore, there must be a technique to read the data to be stored within a table from an external source, such as a card file, and to then store the data within the table.

In addition, it will be noted that the data stored in the tables in the previous programs was specified to be in the proper sequence within the program. There are occasions, however, when an external file is to be used, that the data within the file is stored in one sequence and the sequence of the data required for the program is another sequence. When this occurs, the file of data must be read and sorted into the proper sequence before being loaded into the table. ANSI COBOL makes available for this purpose the SORT verb, which can be used to sort data which is read by the program in order to place it in the desired sequence.

It will also be noted from the programs in Chapter 11 that the routine to perform the table search operation, that is, the initialization and updating of the subscript, the comparison, and the branch back to perform another compare operation must all be written by the programmer. Another feature of ANSI COBOL is the ability to use the SET and SEARCH statements, which will perform the table lookup operation without the need for the programmer to write the entire routine.

The use of the SORT verb and the SET-SEARCH Table Lookup Method will be illustrated in this chapter.

SAMPLE PROGRAM - INPUT

In order to illustrate these concepts a sample program will be used in which input data cards containing the Part Number and Quantity Sold will be used to produce a Sales Report. The format of the input data cards is illustrated below.

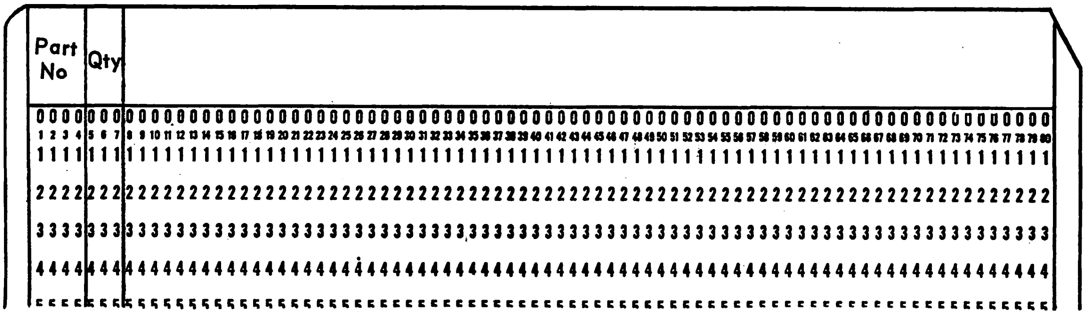
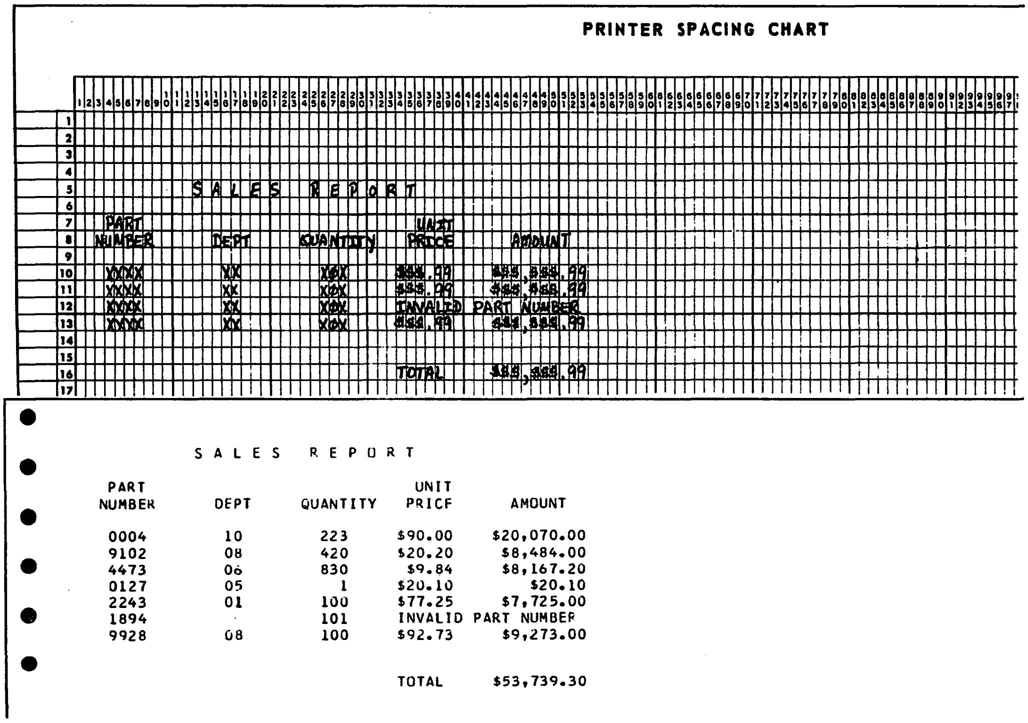


Figure 12-1 Sales Cards

OUTPUT

The output to be produced is to be a Sales Report which contains the Part Number, the Department Number, the Quantity Sold, the Unit Price, and the Total Sales Amount (Quantity X Unit Price). The format of the report is illustrated in Figure 12-2.


PRINTER SPACING CHART
SALES REPORT
PART NUMBER
DEPT
QUANTITY
UNIT PRICE
AMOUNT
XXXX
XX
XXX
$$$.99
$$$,$$$.99
INVALID PART NUMBER
TOTAL

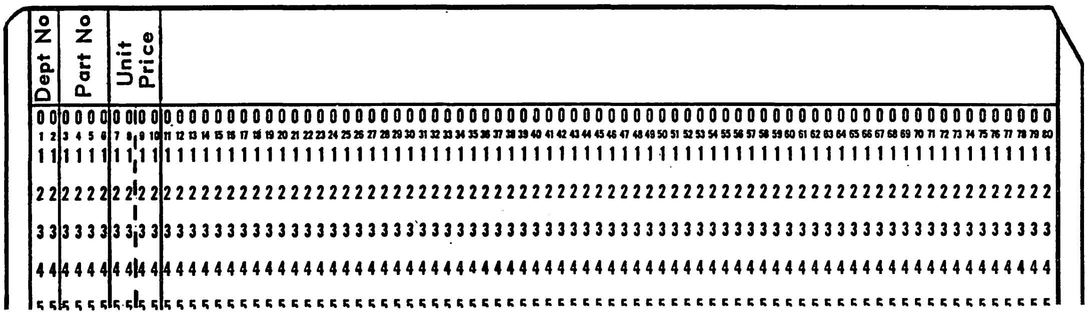
```
                S A L E S   R E P O R T

   PART                                 UNIT
  NUMBER       DEPT      QUANTITY      PRICF        AMOUNT

   0004         10          223       $90.00    $20,070.00
   9102         08          420       $20.20     $8,484.00
   4473         06          830        $9.84     $8,167.20
   0127         05            1       $20.10        $20.10
   2243         01          100       $77.25     $7,725.00
   1894                     101       INVALID PART NUMBEF
   9928         08          100       $92.73     $9,273.00

                                      TOTAL     $53,739.30
```

Figure 12-2 Output Report

Note that the Department Number and the Unit Price are not contained on the input data cards. Both the Department Number and the Unit Price are to be extracted from a table which is to be created from an external file of cards which is to be read into storage.

CARD FILE FOR THE TABLE

The format of the file of cards which are to be used to create the table is illustrated in Figure 12-3. These cards contain a Department Number, a Part Number, and the Unit Price for the part. This file of cards is normally arranged in Department Number Sequence. A portion of the contents of this file is also illustrated in Figure 12-3.



CARD FILE

DEPT	PART NUMBER	UNIT PRICE
01	2243	77.25
05	0127	20.10
06	4473	9.84
08	9102	20.20

SAMPLE DATA

Figure 12-3 Card File and Contents of File

For processing by the program it is necessary to sort this file of cards, which is to be used to create the table, into Part Number sequence rather than Department Number sequence. Thus the processing in the sample program will consist of the following steps:

1. A file of cards comprising the table will be read and stored on a disk work area. The file will then be sorted into Part Number sequence and a table will be built.

2. The input data cards will be read and the Department Number and Unit Price will be extracted from the table based upon the Part Number contained in these cards.

3. The Sales Amount will be calculated by multiplying the Quantity from the cards by the proper Unit Price extracted from the table, and a line will be printed.

SORTING THE CARDS TO BUILD THE TABLE

As noted, the input cards which will be placed in the table are normally stored in a department number sequence. Before they can be placed in the table, they must be sorted into a part number sequence so that the part number in the detail cards can be used to find the unit price and department number through the use of the SEARCH verb. In order to place these cards in the proper sequence, the SORT verb is used. The processing which will be accomplished by the SORT verb and the routines used with the SORT verb are illustrated in the following example.

EXAMPLE

Step 1: The cards are read one at a time and placed in a disk "work file".

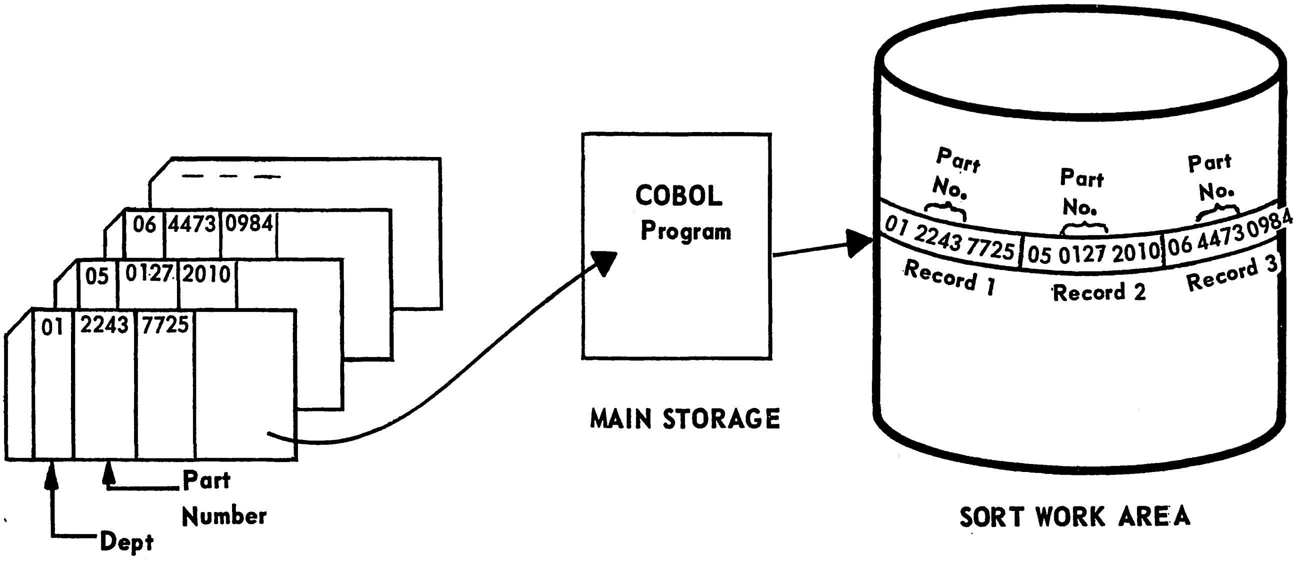

06
4473
0984
05
0127
2010
01
2243
7725
Part Number
Dept
COBOL Program
MAIN STORAGE
Part No.
Part No.
Part No.
01 2243 7725
05 0127 2010
06 4473 0984
Record 1
Record 2
Record 3
SORT WORK AREA

Figure 12-4 Cards are Read and Placed in Sort Work Area

Note from the example above that the cards are read by the program and are stored on a disk device in the same sequence in which they are read. This disk area is the work area which must be defined within the COBOL program for use with the SORT verb. The card input file must also be defined within the COBOL program.

Step 2: The Sort program which is available from the software of the computer system is invoked to sort the records and place them in the sort output area.

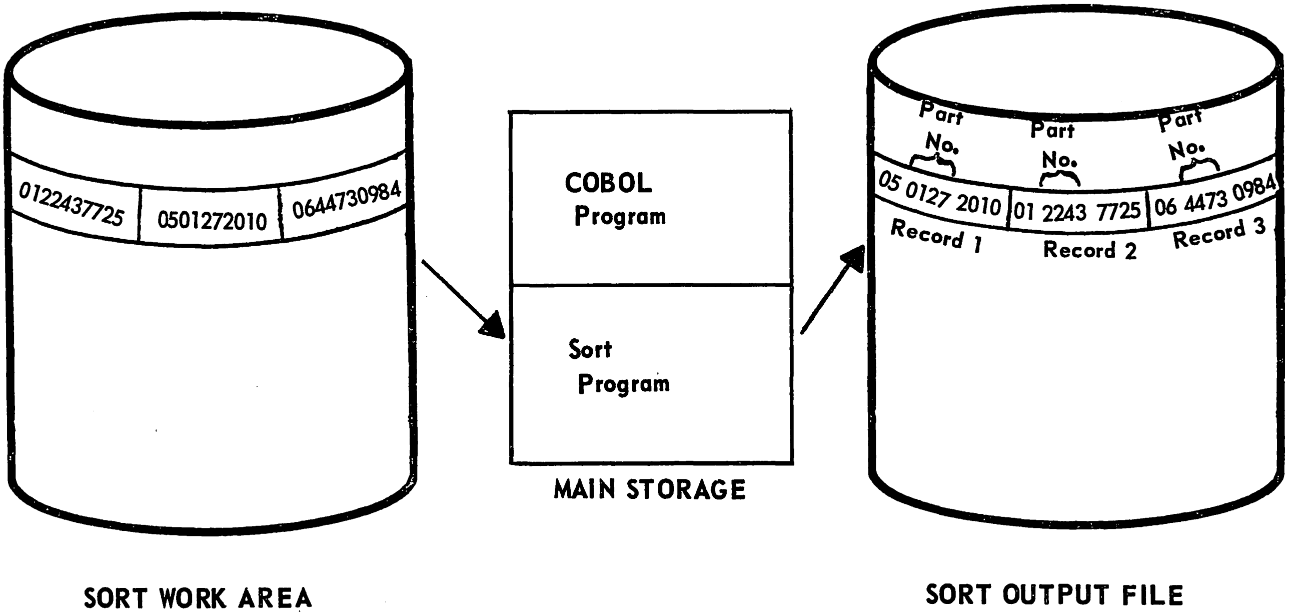

0122437725
0501272010
0644730984
COBOL Program
Sort Program
MAIN STORAGE
Part No.
Part No.
Part No.
05 0127 2010
01 2243 7725
06 4473 0984
Record 1
Record 2
Record 3
SORT WORK AREA
SORT OUTPUT FILE

Figure 12-5 SORT Program Sorts the Data

Note from the example above that the data which is stored in the Sort Work Area from Step 1 is read by the Sort Program and, after being sorted in the proper sequence, is placed in the Sort Output File. This file may then become an input file to the COBOL program so that the data in the proper sequence may be processed.

In the sample program, this sorted data on the disk is to be read by the COBOL program and placed in the table in a part number sequence.

Step 3: Data is read and placed in the table.

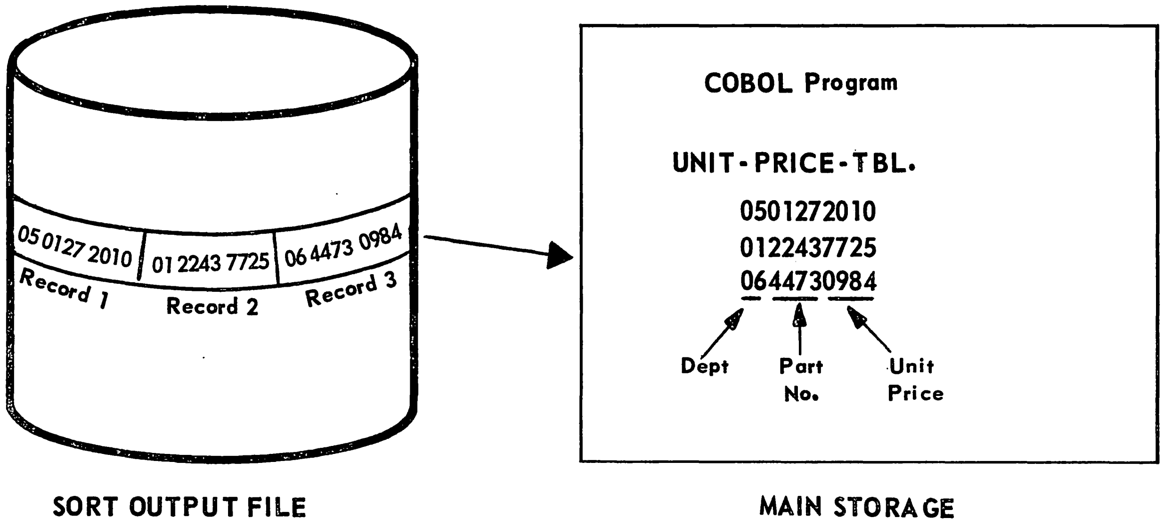

05 0127 2010
01 2243 7725
06 4473 0984
Record 1
Record 2
Record 3
COBOL Program
UNIT-PRICE-TBL.
0501272010
0122437725
0644730984
Dept
Part No.
Unit Price
SORT OUTPUT FILE
MAIN STORAGE

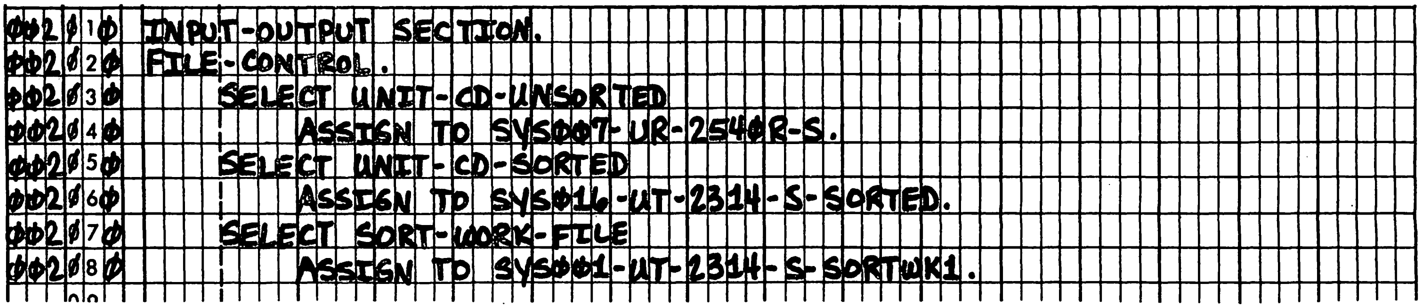
Figure 12-6 Sorted Data is Placed in Table

Note from Figure 12-6 that the sorted data, which is stored on the sort output file is read by the COBOL program and placed in the table UNIT-PRICE-TBL. The data stored in part number sequence within the table is then used when the detail sales cards are read to find both the Unit Price and the Division Number.

ENVIRONMENT DIVISION

In order to use the SORT verb to process the data as illustrated, certain files must be defined in the Environment Division and the Data Division. As can be seen, a card input file (the card containing the data that is to comprise the table), a disk sort work file, and a sort output file must be defined in order to sort the table cards. The example below illustrates the entries used in the sample program to define these three files.

EXAMPLE



```
002010 INPUT-OUTPUT SECTION.
002020 FILE-CONTROL.
002030     SELECT UNIT-CD-UNSORTED
002040         ASSIGN TO SYS007-UR-2540R-S.
002050     SELECT UNIT-CD-SORTED
002060         ASSIGN TO SYS016-UT-2314-S-SORTED.
002070     SELECT SORT-WORK-FILE
002080         ASSIGN TO SYS001-UT-2314-S-SORTWK1.
```

Figure 12-7 Environment Division Entries for Sort Files

The file UNIT-CD-UNSORTED is the file of cards (the table) which is to be read from the card reader. The file is in Department Number sequence. It is this file which is to be sorted.

The file UNIT-CD-SORTED is the output from the sort program, that is, it will contain the records sorted in a part number sequence as illustrated in Figure 12-5. Note that there are several new entries in the Select Statement for the sorted output file. This is because the file is a disk file instead of a unit record file such as a card or printer. The entry SYS016, the symbolic device, is merely a symbolic name which has been assigned to the proper disk drive by the Operating System. As noted in Chapter 2, this entry may or may not be required depending upon the compiler and operating system being utilized.

The class entry of "UT" is used to mean "Utility". This class code is used whenever a sequential file is being defined on a device other than a unit-record device. It is commonly used for sequential files defined on various direct-access devices, such as disk or drums, as well as for files defined for magnetic tape devices.

The device specification is "2314" because the output file is to be on an IBM 2314 Disk Device. It should be noted that this device specification may vary dependent upon the computer and the type of direct-access device being used.

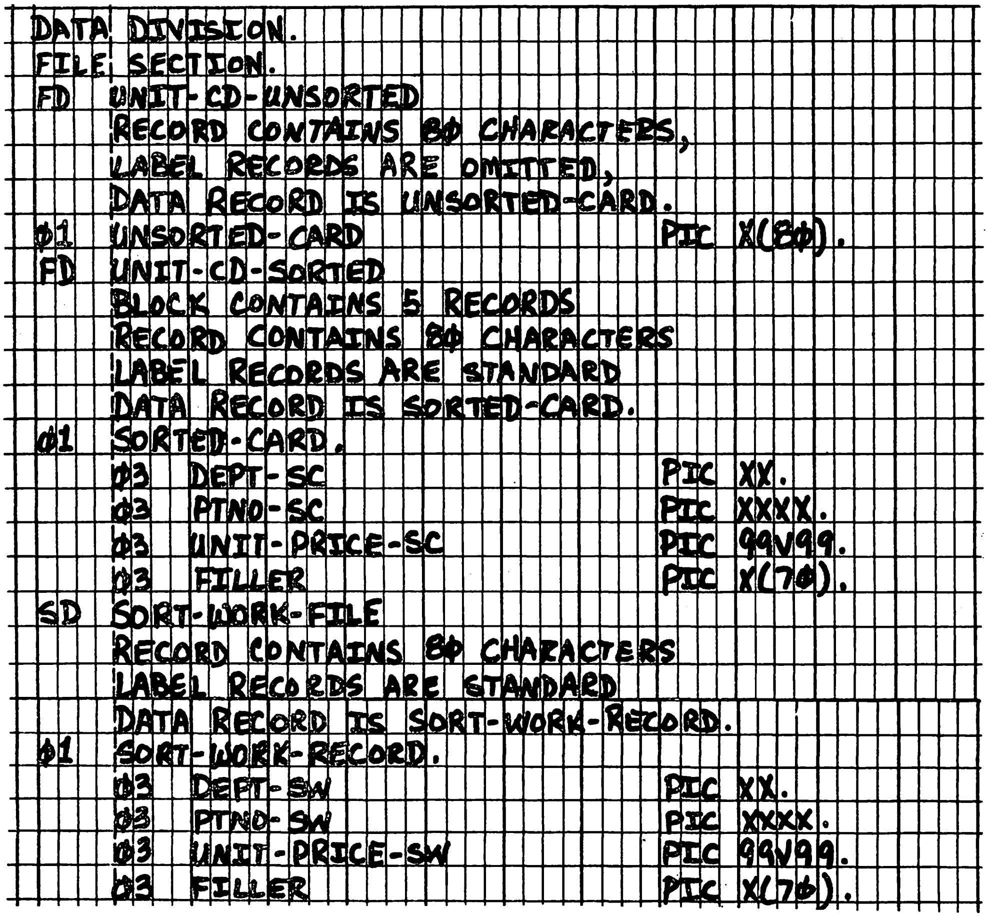
The organization is specified as "S" because the output file is to be a sequential file. The last entry illustrated in Figure 12-7 for the Select Statement is the name "SORTED". This name is used to identify the file to the operating system under which the program will be executed. It is the name which is normally specified in any job control statements associated with the file so that the Operating System can relate the job control card with the file within the program. Whenever labelled files are to be used on disk or tape devices, the Name entry should be included in the SELECT statement.

It will be noted from Figure 12-7 that the sort work file is also to be contained on a 2314 disk device. The symbolic device number and the Name are different than those of the output file, but the remainder of the Select statement is the same. This file will be used by the Sort program as the intermediary storage area for the data to be processed as illustrated in the previous examples.

DATA DIVISION

As with all files defined under COBOL, a file defined in the Environment Division must also have a definition in the Data Division. The Data Division entries for the table card input file, the sort work file, and the sort output file are illustrated below.

EXAMPLE

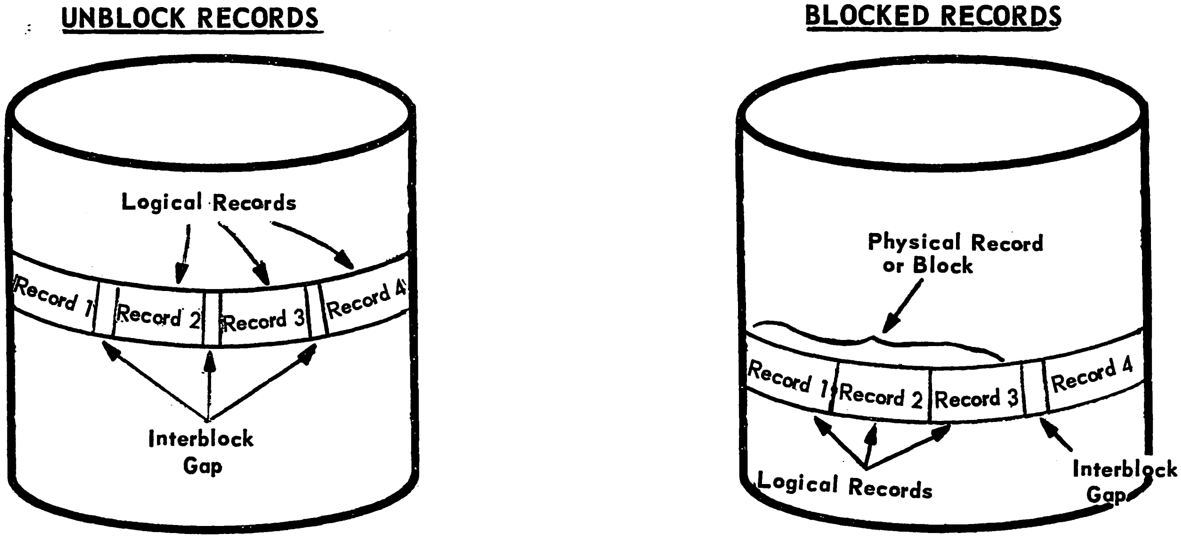


```
DATA DIVISION.
FILE SECTION.
FD  UNIT-CD-UNSORTED
    RECORD CONTAINS 80 CHARACTERS,
    LABEL RECORDS ARE OMITTED,
    DATA RECORD IS UNSORTED-CARD.
01  UNSORTED-CARD                       PIC X(80).
FD  UNIT-CD-SORTED
    BLOCK CONTAINS 5 RECORDS
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS SORTED-CARD.
01  SORTED-CARD.
    03  DEPT-SC                         PIC XX.
    03  PTNO-SC                         PIC XXXX.
    03  UNIT-PRICE-SC                   PIC 99V99.
    03  FILLER                          PIC X(70).
SD  SORT-WORK-FILE
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS SORT-WORK-RECORD.
01  SORT-WORK-RECORD.
    03  DEPT-SW                         PIC XX.
    03  PTNO-SW                         PIC XXXX.
    03  UNIT-PRICE-SW                   PIC 99V99.
    03  FILLER                          PIC X(70).
```

Figure 12-8 Data Division File Definitions

As can be seen in Figure 12-8, the file UNIT-CD-UNSORTED is defined as a regular card input file as in previous examples. The file UNIT-CD-SORTED is the file which is to contain the output data from the sort. As noted previously, it is to be stored on a 2314 disk device. Whenever data from a file is to be stored on a disk or tape device, the records may be stored on a blocked format. Blocking refers to the process of grouping one or more logical records, that is, each individual record which is defined in the data division, into a physical record or block. This concept is illustrated below.

EXAMPLE


UNBLOCK RECORDS

BLOCKED RECORDS

Figure 12-9 Example of Unblocked and Blocked Records

When records are written on a disk file in an unblocked format, each logical record is separated from the next logical record by an interblock gap, which is a space between the records stored on the disk which is required by the disk mechanism in order to be able to identify individual records.

When records are written in a blocked format, logical records are placed next to each other and are written on the disk as a single Physical Record, or Block. These records are "blocked" by the software generated by the COBOL Compiler from the specifications made in the File Section definition of the file. As can be seen, the use of Blocked records will increase the number of records which can be stored in a given space on the disk because there are fewer interblock gaps. In addition, since more than one logical record is read from the disk, that is, each time a read takes place from the disk a physical record is read into main storage, Input/Output operations may take place faster.

From Figure 12-8 it can be seen that the entry "BLOCK CONTAINS 5 RECORDS" is included in the FD statement for the UNIT-CD-SORTED file. This indicates to the compiler that there are to be five logical records contained in each physical record or block which is written for the file. The entry RECORD CONTAINS 80 CHARACTERS specifies that there are 80 characters in each LOGICAL Record.

The entry LABEL RECORDS ARE STANDARD in Figure 12-8 indicates that standard labels are to be used for the disk file. This is normally required when disk files are defined within the program, that is, most operating systems and COBOL compilers require that standard labels are used with disk files. In some operations, however, it may be allowable to have special user-written labels in place of or together with standard labels.

The definition of the sort work file is again illustrated below.

EXAMPLE

```
SD  SORT-WORK-FILE
    RECORD CONTAINS 80 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS SORT-WORK-RECORD.
01  SORT-WORK-RECORD.
    03  DEPT-SW                  PIC XX.
    03  PTNO-SW                  PIC XXXX.
    03  UNIT-PRICE-SW            PIC 99V99.
    03  FILLER                   PIC X(70).
```

Figure 12-10 Definition of Sort Work File

Note from the example above that the letters "SD" are placed in Area A instead of the more familiar letters "FD". The SD definition indicates that the file being defined is merely to give the attributes of a file which is automatically defined by the SORT program. Thus, the SD file is not actually a new file which is unique to the COBOL program, but rather a statement of the attributes which a file defined within the SORT program is to assume.

As can be seen from Figure 12-10, the remaining entries for a SD file are the same as those used in previous examples to define the attributes of a file. It should be noted that the file, as defined in the SELECT statement in Figure 12-7 is to be stored on a direct-access device but that the BLOCK CONTAINS . . . entry is not included. This is because the Sort program will use its own blocking factor depending upon the format of the records to be sorted and, therefore, this entry may not be included in the definition of the Sort Work File.

The general format for the SD entry is illustrated in Figure 12-11.

```
SD sort-file-name

          [DATA  { RECORD IS    }  data-name-1 [data-name-2] ...]
                 { RECORDS ARE  }

          [RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]

          [LABEL { RECORD IS    }  { STANDARD }  ].
                 { RECORDS ARE  }  { OMITTED  }
```

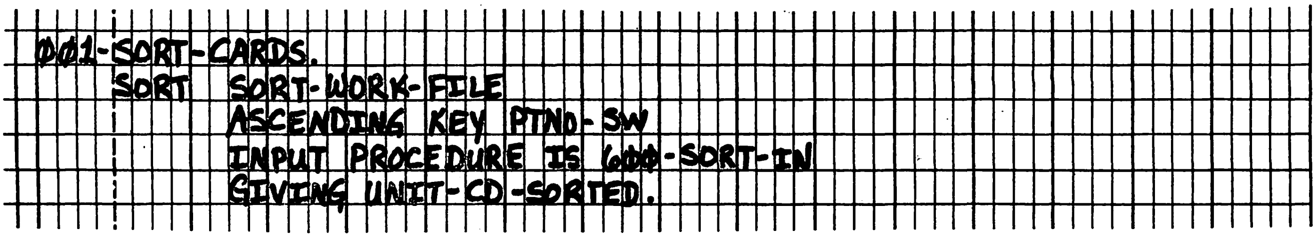
Figure 12-11 General Format – SD Statement

It can also be seen in Figure 12-10 that the fields within the records to be sorted are defined. The names within the record may be used in the SORT statement which is used in the Procedure Division.

SORT STATEMENT

As will be noted from Figure 12-4, the first step which must be accomplished is the reading of the table cards by the COBOL program, sorting these cards, and placing them in the sort output file. This is accomplished through the use of the SORT verb and the accompanying routines. The SORT verb used in the sample program is illustrated in Figure 12-12.

EXAMPLE



```
001-SORT-CARDS.
    SORT  SORT-WORK-FILE
          ASCENDING KEY PTNO-SW
          INPUT PROCEDURE IS 600-SORT-IN
          GIVING UNIT-CD-SORTED.
```

Figure 12-12 Example of SORT Verb

The general format of the SORT verb is illustrated below.

```
                        { DESCENDING }
SORT file-name-1 ON     {            } KEY {data-name-1} ...
                        { ASCENDING  }

                { DESCENDING }
          [ON   {            }   KEY {data-name-2} ...] ...
                { ASCENDING  }

         { INPUT PROCEDURE IS section-name-1 [THRU section-name-2] }
         { USING file-name-2                                       }

         { OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4] }
         { GIVING file-name-3                                       }
```

Figure 12-13 General Format of SORT Verb

Note that the first statement in the sentence to be used to define the sorting procedure is the word SORT, which is required. The entry "file-name-1", which is also required, is the name of the file defined by the SD Entry in the Data Division. Thus, from Figure 12-12 it can be seen that the filename specified is SORT-WORK-FILE, which is the file defined in Figure 12-10.

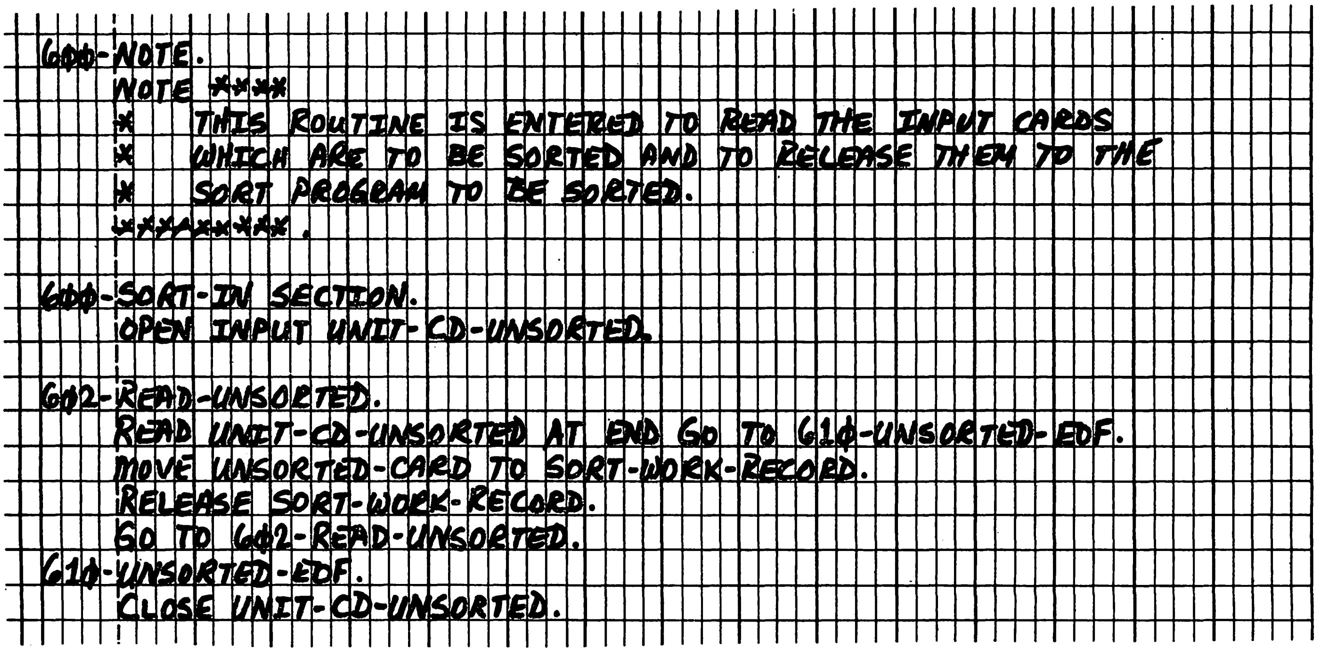
The next entry is the optional word ON followed by one of the required words, either ASCENDING or DESCENDING. This word is used to indicate the sequence in which the particular field specified by data-name-1 is to be sorted. The word ASCENDING specifies that the data is to be sorted in an ascending sequence, as illustrated in Figure 12-5. It should be noted that most Sort programs will also allow data to be sorted in a descending sequence, and this is specified for the word DESCENDING.

As noted, the data-name entry following either Ascending or Descending specifies the data name of the control field in the record as defined in the File Section for the SD file. In the example in Figure 12-12, this field is PTNO-SW because the records are to be sorted on part number. It should be noted also that more than one field may be specified as a control field by using the data-name-2, data-name-3, etc. entries. The fields specified for data-name-2, etc. are minor to the previous data-name, that is, the field specified first with data-name-1 is the major field in the sort, the field specified second with data-name-2 is the next major field in the sort, etc. Different sort programs for different machines have varying maximums concerning the number of fields which may be specified.

The entries described previously have specified the file to be sorted and the method in which the records are to be sorted. It remains to specify the manner in which the records are to be read and placed on the work file and also the manner in which the records are to be placed on the sort output file. In general, this may be done in one of two ways – the COBOL program issues the READ Command to read the file or the WRITE Command to write the output file, or the programmer can allow routines generated as a result of the SORT verb to handle the input and output processing.

In the sample program, the input data cards are to be read by a special Section within the program and the records, after being sorted, are to be written on the output file by the routine generated as a result of the SORT verb. In order to indicate that a special routine within the program is to be used to read the input data, the entry "INPUT PROCEDURE IS" is specified within the SORT statement as illustrated in Figure 12-12. The section-name-1 entry must be a section which is defined within the program. The section used in the sample program is illustrated in Figure 12-14.

EXAMPLE



```
600-NOTE.
    NOTE ****
    *   THIS ROUTINE IS ENTERED TO READ THE INPUT CARDS
    *   WHICH ARE TO BE SORTED AND TO RELEASE THEM TO THE
    *   SORT PROGRAM TO BE SORTED.
    *********.

600-SORT-IN SECTION.
    OPEN INPUT UNIT-CD-UNSORTED.

602-READ-UNSORTED.
    READ UNIT-CD-UNSORTED AT END GO TO 610-UNSORTED-EOF.
    MOVE UNSORTED-CARD TO SORT-WORK-RECORD.
    RELEASE SORT-WORK-RECORD.
    GO TO 602-READ-UNSORTED.
610-UNSORTED-EOF.
    CLOSE UNIT-CD-UNSORTED.
```

Figure 12-14 INPUT Section

Note in the example in Figure 12-14 that following the note the label "600-SORT-IN SECTION" is placed in the source program. This label is used to define a SECTION within the source program. A Section may be thought of much the same as a procedure paragraph except that a Section consists of one or more procedure paragraphs where a procedure paragraph consists of one or more sentences. Thus, a section is more encompassing than a procedure paragraph. When stating the input routine to be used to read data, a Section must be stated in the Sort verb (see Figure 12-12) and must be defined in the program as illustrated above.

As a result of the statement in the Sort verb, the 600-SORT-IN Section is essentially performed as if a perform statement had been specified. Within the Input Section, the sort input file, UNIT-CD-UNSORTED, is read, and the card is moved from the card input area to the sort work file output area. It should be noted in this example that the data is merely moved from the input area to the sort work output area. It is possible, however, that any type of processing which is required could be performed on the data at this point prior to placing the data on the sort work file. Thus, one of the advantages of using a routine within the program to read the input data and write it on the sort work file is that the input data may be processed prior to being written. When the routines generated by the SORT verb are used to write the work file, as will be shown, the program has no opportunity to process the input data.

After the input data has been read and placed in the work file output area, the RELEASE verb is used to place the data on the work file. The format notation for the RELEASE verb is illustrated in Figure 12-15.

Format Notation

```
RELEASE sort-record-name [FROM identifier]
```

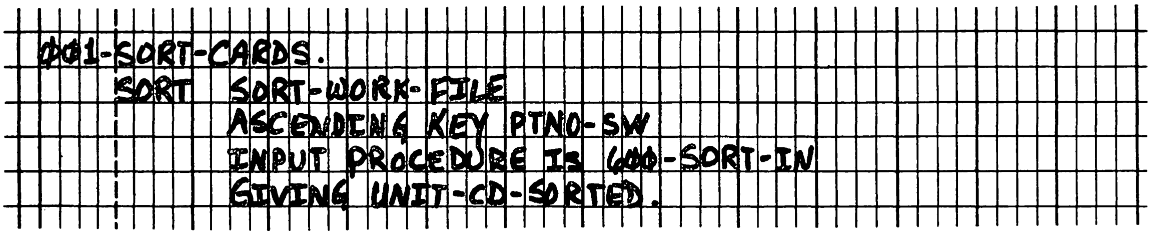
Figure 12-15 Format Notation – RELEASE Statement

From the format notation in Figure 12-15 it can be seen that the Release Statement is used in essentially the same manner as the WRITE statement, that is, the name of the output area is specified following the RELEASE verb. Optionally, the FROM specification can be made in the same manner as with the Write statement. The output area specified must be the name which is defined within the SD statement in the File Section of the Data Division (see Figure 12-8).

After the Release statement is executed to place the record on the work file, control is transferred back to the READ statement to read the next input record. When all of the records have been read and "released", the input file is closed and control then returns to the coding generated by the SORT verb because the section is complete.

The records are then sorted because all of the records have been placed on the sort work file. After they are sorted, the Sort routine must be directed to place the sorted records on the sort output file. In the sample program, this is accomplished by the "GIVING" entry in the SORT routine as illustrated below.

EXAMPLE



```
001-SORT-CARDS.
    SORT SORT-WORK-FILE
         ASCENDING KEY PTNO-SW
         INPUT PROCEDURE IS 600-SORT-IN
         GIVING UNIT-CD-SORTED.
```

Figure 12-16 SORT Statement

As can be seen, the entry "GIVING UNIT-CD-SORTED" is included in the Sort statement following the Input Procedure definition. This statement says that the sorted output data is to be stored on the file UNIT-CD-SORTED. It will be recalled that this file is defined in the Data Division (see Figure 12-8). Thus, this entry specifies to the Sort program that once the data is sorted, it is to be stored on the file defined following the GIVING word. It should be noted that the data will be written directly on the output file by the routines generated from the SORT verb without any intervention from any of the routines in the COBOL program.

USING Clause

As was noted previously, it is possible for the input data to be read by the routines generated as a result of the Sort statement instead of requiring that an input section be written by the COBOL programmer. This is accomplished through the use of the USING clause in the Sort statement. If it was desired to use this technique in the sample program, the following Sort statement could be used.

EXAMPLE

```
001-SORT-CARDS.
   SORT    SORT-WORK-FILE
           ASCENDING KEY PTNO-SW
           USING UNIT-CD-UNSORTED
           GIVING UNIT-CD-SORTED.
```

Figure 12-17 Example of USING Clause

Note in the example in Figure 12-17 that instead of specifying an Input Procedure to be used to read the input data cards which are to be sorted, the USING clause is specified followed by the filename of the unsorted card file. It will be recalled that in the previous examples, a procedure was used to read the card file. In this example, the routines generated by the SORT verb will automatically read the card input file and will place the cards in the sort work file. Thus, the input section, the READ statement within the section, and the RELEASE verb need not be used when the USING clause is included in the SORT sentence.

OUTPUT PROCEDURE

In the example used in the sample program, the GIVING option is used in the Sort statement to cause the sorted records to be placed in an output file. An alternative method is to retrieve the records directly from the sort work file after they have been sorted. In order to do this, the OUTPUT PROCEDURE entry in the Sort Statement is utilized together with an output procedure which is written in the COBOL program. The example in Figure 12-18 illustrates an output procedure which could be used together with the required Sort statement.

EXAMPLE

```
001-SORT-CARDS.
    SORT  SORT-WORK-FILE
          ASCENDING KEY PTNO-SW
          USING UNIT-CD-SORTED
          OUTPUT PROCEDURE IS 700-GET-SORTED.
```

```
700-NOTE.
    NOTE ****
    *     THIS ROUTINE IS ENTERED TO RETRIEVE THE SORTED
    *     RECORDS FROM THE SORT WORK FILE AND PRINT THEM
    *     ON THE PRINTER.
    *********.

700-GET-SORTED SECTION.
    OPEN OUTPUT PRINT-FILE.

702-READ.
    RETURN SORT-WORK-FILE INTO PRINT-AREA
          AT END GO TO 705-DONE.
    WRITE PRINT-IO FROM PRINT-AREA
          AFTER ADVANCING 1 LINES.
    GO TO 702-READ.
705-DONE. EXIT.
```

Figure 12-18 Example of OUTPUT PROCEDURE

In the example in Figure 12-18 it can be seen that an output procedure is specified within the SORT statement and the procedure is named 700-GET-SORTED. It will be noted that the format for stating that an output procedure is to be used is the same as the format used for stating an input procedure is to be used (see Figure 12-13).

The output procedure must be a section in the same way as the input procedure must be. After the printer output file is opened, the RETURN verb is used to make the sorted record on the sort work file available. The general format of the RETURN statement is illustrated in Figure 12-19.

```
RETURN sort-file-name RECORD [INTO identifier]
     AT END irperative-statement
```

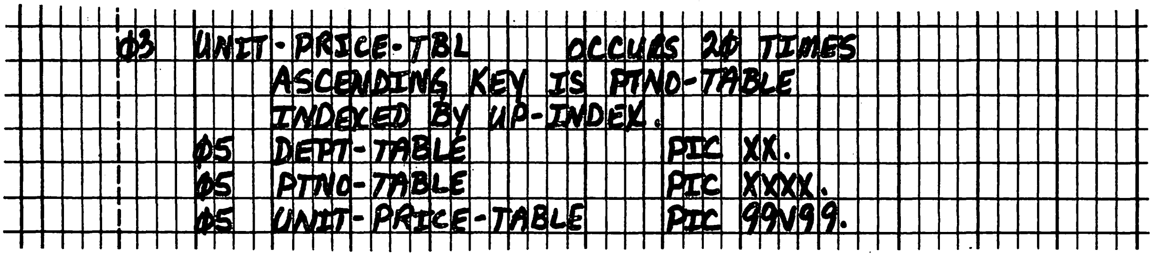
Figure 12-19 Format Notation – RETURN Statement

The sort-file-name required for the Return statement is the file which is defined by the SD entry in the Data Division. The optional INTO Clause is used in the same manner as the INTO clause used with the Read statement. Thus, in Figure 12-18, it can be seen that the sorted record will be placed in the PRINT-AREA defined within the Working-Storage Section of the Data Division. The AT END statement is used in the same manner as with the Read statement of previous programs.

TABLE DEFINITION

As noted previously, after the table cards are resorted into a part number sequence, they are to be placed into a table. The definition of the table used in the sample program is illustrated below.

EXAMPLE



```
       03  UNIT-PRICE-TBL      OCCURS 20 TIMES
               ASCENDING KEY IS PTNO-TABLE
               INDEXED BY UP-INDEX.
           05  DEPT-TABLE              PIC XX.
           05  PTNO-TABLE              PIC XXXX.
           05  UNIT-PRICE-TABLE        PIC 99V99.
```

Figure 12-20 Definition of TABLE

Note from the example above that the OCCURS Clause is used for this table definition in the same manner as used for the tables in Chapter 11. Two additional entries have been included, however, because the SEARCH verb is to be used to perform the table lookup operation. These are the ASCENDING KEY clause and the INDEXED BY clause.

The ASCENDING KEY clause is used merely to notify the compiler which field within the table is to be the control field, that is, which field is to be used in the table lookup operation. Since the Part Number is the field which is to be in an ascending sequence, and which will be the control field, the field PTNO-TABLE is specified as the ascending key field.

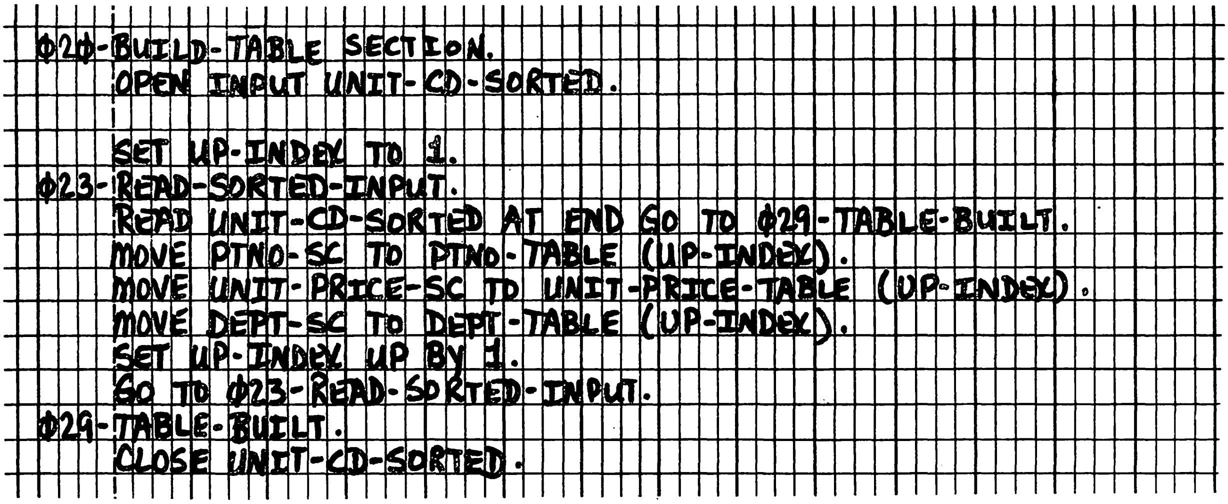
The INDEXED BY clause is used to specify the name of a field which will be used as the subscript when referencing the various elements within the table. In previous examples, it will be recalled that the INDEXED BY clause was not utilized and the field to be used as a subscript was defined by a level 77 entry in Working-Storage. When the INDEXED BY clause is used, the data-name specified by the programmer gives a name to a field which is automatically generated by the COBOL compiler and which need not be defined by the programmer. Thus, the field UP-INDEX will be generated by the compiler and will be used as the subscript to reference elements within the table.

The fields within the table are defined as illustrated previously for the department number, the part number, and the unit price.

LOADING THE TABLE

After the data has been sorted through the use of the SORT verb, it must be read from the sort output file and placed in the table defined in Figure 12-20. The entries from the sample program used to load the table are illustrated in Figure 12-21.

EXAMPLE



```
020-BUILD-TABLE SECTION.
    OPEN INPUT UNIT-CD-SORTED.

    SET UP-INDEX TO 1.
023-READ-SORTED-INPUT.
    READ UNIT-CD-SORTED AT END GO TO 029-TABLE-BUILT.
    MOVE PTNO-SC TO PTNO-TABLE (UP-INDEX).
    MOVE UNIT-PRICE-SC TO UNIT-PRICE-TABLE (UP-INDEX).
    MOVE DEPT-SC TO DEPT-TABLE (UP-INDEX).
    SET UP-INDEX UP BY 1.
    GO TO 023-READ-SORTED-INPUT.
029-TABLE-BUILT.
    CLOSE UNIT-CD-SORTED.
```

Figure 12-21 Routine to Load Table

Note from the example above that the file which contains the sorted records, UNIT-CD-SORTED, is opened as an input file so that it can be read and the records placed in the table. The SET statement is then used to initialize the Index value to "1". The SET statement must be used whenever the index defined in the Data Division is to be set to any value or is to be incremented or decremented. The general format of the SET statement to initialize the value in an index field is illustrated in Figure 12-22.

```
     { index-name-1 [index-name-2]... }      { index-name-3 }
SET  {                                }  TO  { identifier-3 }
     { identifier-1 [identifier-2]... }      { literal-1    }
```

Figure 12-22 Format Notation – SET Statement

Note that the word SET is required. The value for index-name-1 in the example in Figure 12-21 is the name UP-INDEX, which is the index value defined in the Data Division (see Figure 12-20). The word TO is required and then there may be contained in the statement either another index name, the identifier of a field which must contain numeric elementary items described as integers, or a literal with a positive value. Note from Figure 12-21 that the literal "1" is used to initialize the index value to "1", which will cause the first element in the table to be referenced when UP-INDEX is used as a subscript.

After the index is initialized, a record is read from the sorted data in the file UNIT-CD-SORTED. This is the disk file which contains the sorted table records. The part number, unit price, and department number are then moved to the first element in the table.

The index is then incremented by one so that the next pass of the routine will reference the second element of the table. The format of the SET statement used to increment the index is illustrated below.

```
                                            ( UP BY   )  ( identifier-4 )
SET index-name-4 [index-name-5] ...        {           }{                }
                                            ( DOWN BY )  ( literal-2    )
```

Figure 12-23 Format of SET Statement

Note from Figure 12-23 that the word SET is required and that the index-name-4 entry must be the name of the index area to be incremented or decremented. The next words, either UP BY or DOWN BY, are required and specify that the value in the index area is to be incremented or decremented, respectively.

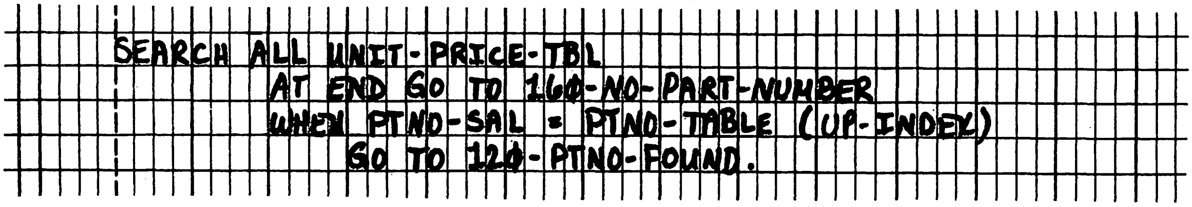
The value in the area specified by identifier-4 or the literal-2 is the value used to determine how much the index is incremented or decremented. In the example in Figure 12-21, it can be seen that the index area UP-INDEX is incremented by the value 1. Thus, on the second pass of the routine, the value in UP-INDEX will be 2, not 1. This processing will continue until the entire file has been read, that is, until all of the records have been read from the sorted input file and the elements in the table loaded. At that time, control will pass to the routine (029-TABLE-BUILT) which will read the detail data, process it against the table, and create the printed report.

SEARCH ALL STATEMENT

After the table has been loaded, the detail processing can begin. The basic steps are to initialize the index value, read the detail input cards, perform the table lookup operation and extract the department number and unit price from the table, and then calculate the amount by multiplying the quantity times the unit price.

In order to perform the table lookup operation, the SEARCH ALL statement is used. The SEARCH ALL statement used in the sample program is illustrated in Figure 12-24.

EXAMPLE



```
SEARCH ALL UNIT-PRICE-TBL
    AT END GO TO 160-NO-PART-NUMBER
    WHEN PTNO-SAL = PTNO-TABLE (UP-INDEX)
      GO TO 120-PTNO-FOUND.
```

Figure 12-24 Example of SEARCH ALL Statement

The format notation of the SEARCH ALL statement is illustrated below.

```
SEARCH ALL identifier-1 [AT END imperative-statement-1]

                        { imperative-statement-2 }
    WHEN condition-1    {                        }
                        { NEXT SENTENCE          }
```

Figure 12-25 Format Notation – SEARCH ALL Statement

The SEARCH ALL statement is used to perform the table search in order to find the part number in the table which corresponds to the part number in the input detail card (PTNO-SAL). The words SEARCH ALL are required when the entire table is to be searched. It should be noted also that when the entire table is to be searched using the SEARCH ALL statement, the table must be organized so that the control field in the table (PTNO-TABLE) is in either an ascending or descending sequence, as specified in the table definition (see Figure 12-20).

Identifier-1 in the format notation in Figure 12-25 must be the name of the table as defined in the Data Division. In the example in Figure 12-24, the name of the table is UNIT-PRICE-TBL (see Figure 12-20). The "AT END" portion of the SEARCH ALL statement specifies the processing which is to occur if the entire table is searched and the condition specified is not found. In the example the routine 160-NO-PART-NUMBER is entered if the condition is not satisfied so that an error message can be written on the report. It should be noted that any imperative statements could be specified in the same manner as in the AT END portion of the READ statement.

The word WHEN is required in the SEARCH ALL statement to indicate that the statements following are to specify the conditions which are to be satisfied in order to successfully complete the table lookup operation. The "condition-1" entry should be an expression which uses the "equal to" conditioned operator. In the sample program it is desired to extract the department and unit price from the table when a match is found between the part number in the input detail card and a part number in the table. Therefore, the condition specified is PTNO-SAL = PTNO-TABLE (UP-INDEX). When this condition occurs, that is, when the part number in the detail card, PTNO-SAL, is equal to the part number in the table, PTNO-TABLE (UP-INDEX), then the imperative sentence specified is executed. In the example, the GO TO 120-PTNO-FOUND statement will be executed..

It should be noted that the entire table lookup operation is performed by the SEARCH ALL statement. Thus, the index value UP-INDEX is automatically incremented or decremented as required for the operation, the comparison specified in the condition-1 statement is automatically performed, and the imperative-statement-2 will be performed when the condition is satisfied. Thus, through the use of the SEARCH ALL statement, the table lookup routine is performed without the need of the programmer writing the routine to update the subscript, perform the comparison, and so on.

SEARCH STATEMENT

In the sample program, the use of the SEARCH ALL statement is illustrated. It should be noted, however, that there is another format of the SEARCH statement and this format is illustrated below.

```
                                    ( index-name-1 )
SEARCH identifier-1 [VARYING       <               > ]
                                    ( identifier-2 )

    [AT END imperative-statement-1]

                          ( imperative-statement-2 )
    WHEN conditicn-1     <                          >
                          ( NEXT SENTENCE           )

                          ( imperative-statement-3 )
    [WHEN condition-2    <                          > ]...
                          ( NEXT SENTENCE           )
```

Figure 12-26 Format Notation – SEARCH Statement

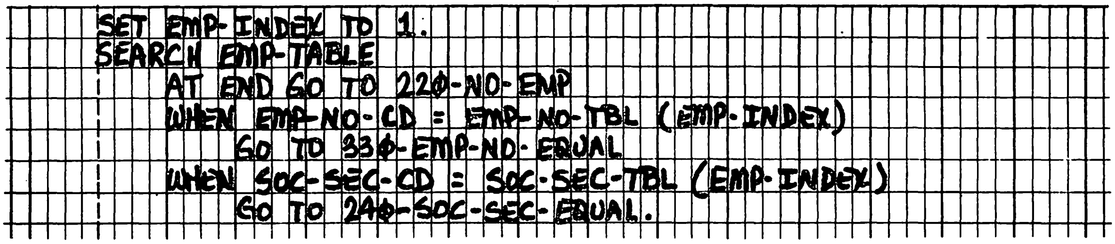
Note from the format notation that only the word SEARCH is specified when this format is to be used. Identifier-1 is the name of the table which is to be searched. The optional VARYING option may be included if the index specified is one other than the index which is specified when the table is defined in the Data Division by the INDEXED BY statement. The Index is varied to check each entry within the table in a sequential manner.

The "AT END" entry is utilized in the same manner as with the SEARCH ALL entry, that is, it indicates the processing to be done if one of the conditions is not satisfied.

It will be noted from the format notation illustrated in Figure 12-26 that more than one condition may be specified when the SEARCH verb is used. The condition specified may be one for class tests, sign tests, or any of the relation conditions. When more than one condition is specified, the table lookup process will be terminated successfully when any one of the conditions is found. Thus, the conditions act in an "OR" relationship, not an "AND" relationship, that is, when the first condition, or the second condition, or the third condition, etc. is satisfied, the table lookup process is complete.

The following is an example of the use of the SEARCH statement.

EXAMPLE



```
SET EMP-INDEX TO 1.
SEARCH EMP-TABLE
   AT END GO TO 220-NO-EMP
   WHEN EMP-NO-CD = EMP-NO-TBL (EMP-INDEX)
      GO TO 330-EMP-NO-EQUAL
   WHEN SOC-SEC-CD = SOC-SEC-TBL (EMP-INDEX)
      GO TO 240-SOC-SEC-EQUAL.
```

Figure 12-27 Example of SEARCH Statement

In the example above, the EMP-TABLE is to be searched. If none of the conditions specified are found, the 220-NO-EMP routine will be entered. As can be seen, there are two conditions which are specified. If the field EMP-NO-CD is equal to the field EMP-NO-TBL (EMP-INDEX), then the 330-EMP-NO-EQUAL routine is entered. If the SOC-SEC-CD field is equal to the SOC-SEC-TBL (EMP-INDEX) field, then the 240-SOC-SEC-EQUAL routine is entered. Note that if either of these conditions is satisfied, the table search routine is terminated and the specified routine is entered. If both of them are true, then the condition specified first is treated as true, that is, if both the employee numbers are equal and the social security numbers are equal, the 330-EMP-NO-EQUAL routine will be entered because it is checked before the social security numbers.

This chapter has illustrated the use of the SORT verb and the SEARCH and SEARCH ALL statements. The SORT verb is useful when data must be in a prescribed sequence prior to processing by the program. It is an alternative to using the SORT Utility program available with most computers and in many instances, will prove more efficient.

The SEARCH statements normally perform the table look-up operation in a more efficient manner than when the table look-up routine is written within the program, such as in Chapter 11. The SEARCH ALL routine is especially efficient when the table can be placed in an ascending or descending sequence because the "binary" look-up technique is used and this normally leads to a faster table search operation.

SAMPLE PROGRAM

The sample program illustrated in this chapter contains an application of both the SORT statement and the SEARCH ALL statement. As noted previously, the file of cards must be read to build the table in a part number sequence, then the sales detail cards are read and processed against this table. A Sales Report is produced.

INPUT

Input consists of cards which are used to build the Unit Price Table and Sales Cards which are processed against the table. The format of these cards is illustrated below.

TABLE CARDS

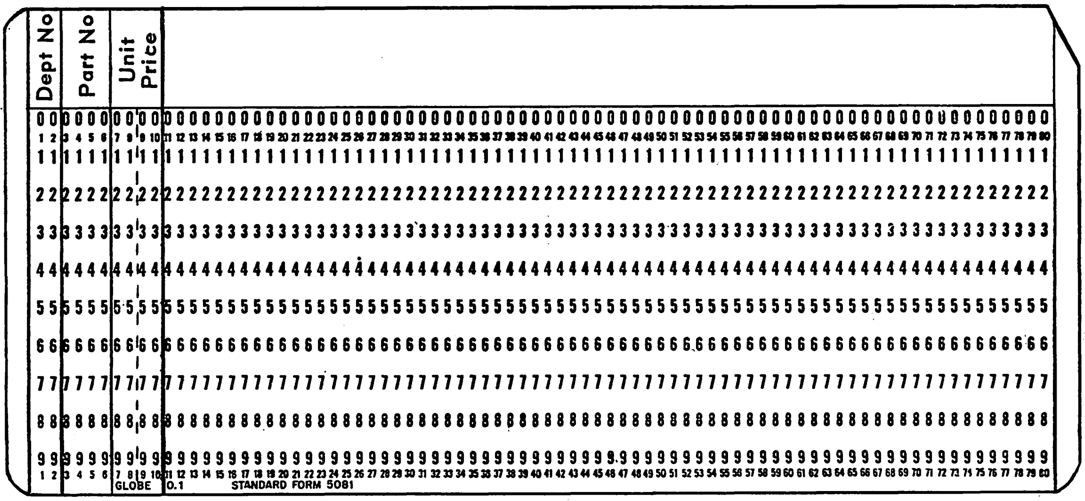

Dept No
Part No
Unit Price
GLOBE NO.1 STANDARD FORM 5081

Figure 12-28 Table Cards

SALES CARDS

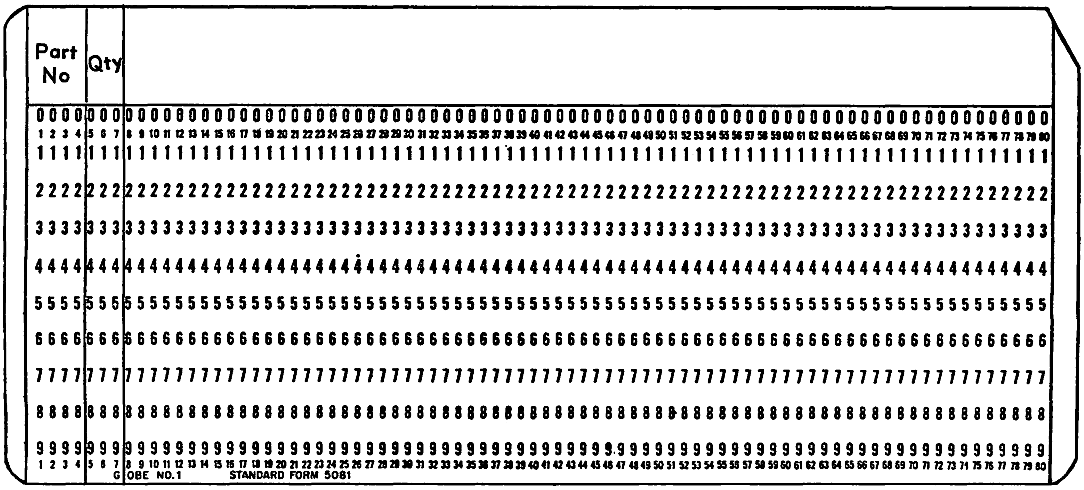
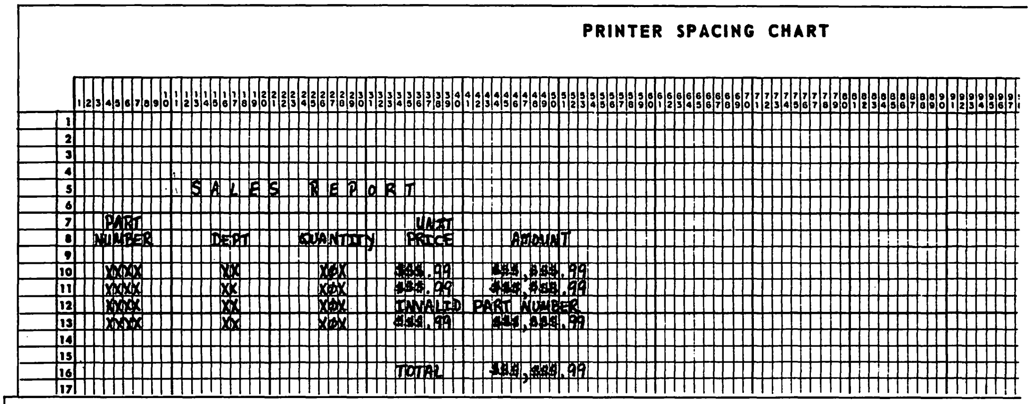

Part No
Qty
GLOBE NO.1 STANDARD FORM 5081

Figure 12-29 Sales Cards

OUTPUT

The final output of the program consists of a Sales Report. The format is illustrated below.



PRINTER SPACING CHART

```
            S A L E S   R E P O R T

  PART                              UNIT
 NUMBER        DEPT     QUANTITY    PRICE       AMOUNT

  XXXX          XX        X0X      $$$.99     $$$,$$$.99
  XXXX          XX        X0X      $$$.99     $$$,$$$.99
  XXXX          XX        X0X      INVALID  PART NUMBER
  XXXX          XX        X0X      $$$.99     $$$,$$$.99


                                   TOTAL      $$$,$$$.99
```

```
            S A L E S   R E P O R T

  PART                              UNIT
 NUMBER        DEPT     QUANTITY    PRICE        AMOUNT

  0004          10         223     $90.00     $20,070.00
  9102          08         420     $20.20      $8,484.00
  4473          06         830      $9.84      $8,167.20
  0127          05           1     $20.10         $20.10
  2243          01         100     $77.25      $7,725.00
  1894                     101     INVALID  PART NUMBER
  9928          08         100     $92.73      $9,273.00

                                   TOTAL      $53,739.30
```

Figure 12-30 Sales Report

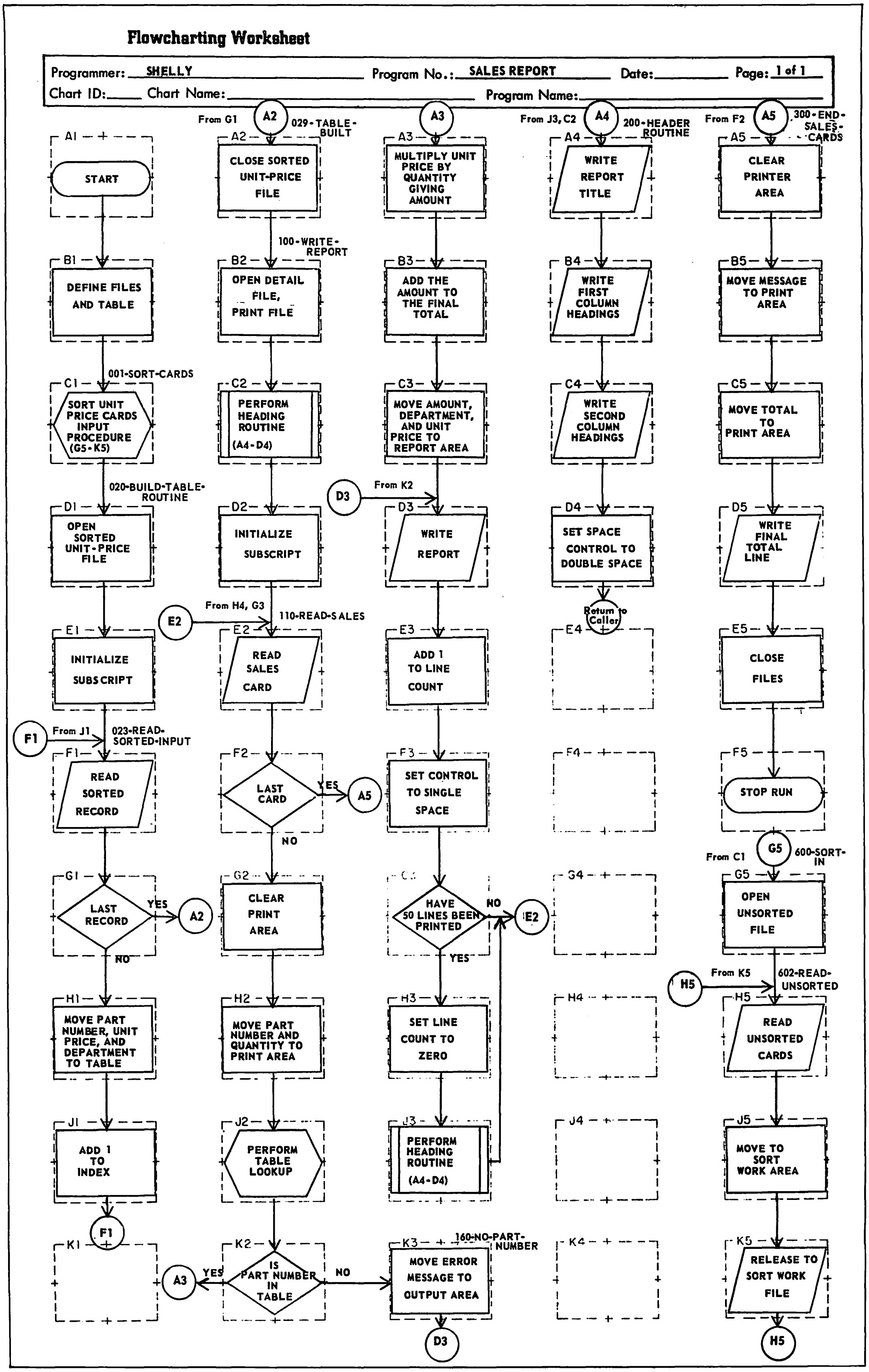
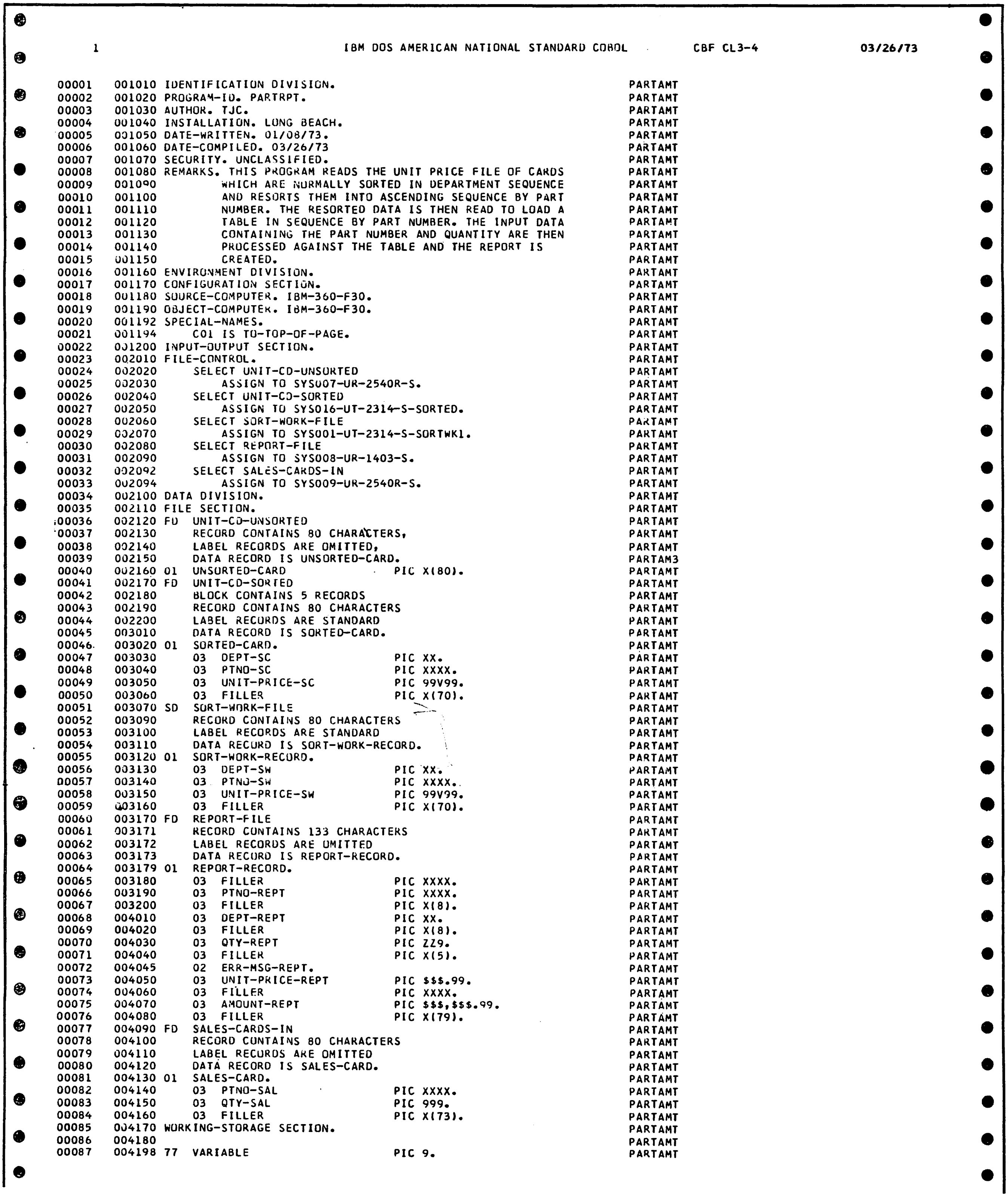

Flowcharting Worksheet
Programmer: SHELLY
Program No.: SALES REPORT
Date:
Page: 1 of 1
Chart ID:
Chart Name:
Program Name:
A1
START
B1
DEFINE FILES AND TABLE
001-SORT-CARDS
C1
SORT UNIT PRICE CARDS INPUT PROCEDURE (G5-K5)
020-BUILD-TABLE-ROUTINE
D1
OPEN SORTED UNIT-PRICE FILE
E1
INITIALIZE SUBSCRIPT
F1
From J1
023-READ-SORTED-INPUT
F1
READ SORTED RECORD
G1
LAST RECORD
YES
A2
NO
H1
MOVE PART NUMBER, UNIT PRICE, AND DEPARTMENT TO TABLE
J1
ADD 1 TO INDEX
F1
K1
From G1
A2
029-TABLE-BUILT
A2
CLOSE SORTED UNIT-PRICE FILE
100-WRITE-REPORT
B2
OPEN DETAIL FILE, PRINT FILE
C2
PERFORM HEADING ROUTINE (A4-D4)
D2
INITIALIZE SUBSCRIPT
E2
From H4, G3
110-READ-SALES
E2
READ SALES CARD
F2
LAST CARD
YES
A5
NO
G2
CLEAR PRINT AREA
H2
MOVE PART NUMBER AND QUANTITY TO PRINT AREA
J2
PERFORM TABLE LOOKUP
K2
A3
YES
IS PART NUMBER IN TABLE
NO
A3
A3
MULTIPLY UNIT PRICE BY QUANTITY GIVING AMOUNT
B3
ADD THE AMOUNT TO THE FINAL TOTAL
C3
MOVE AMOUNT, DEPARTMENT, AND UNIT PRICE TO REPORT AREA
D3
From K2
D3
WRITE REPORT
E3
ADD 1 TO LINE COUNT
F3
SET CONTROL TO SINGLE SPACE
G3
HAVE 50 LINES BEEN PRINTED
NO
E2
YES
H3
SET LINE COUNT TO ZERO
J3
PERFORM HEADING ROUTINE (A4-D4)
K3
160-NO-PART-NUMBER
MOVE ERROR MESSAGE TO OUTPUT AREA
D3
From J3, C2
A4
200-HEADER ROUTINE
A4
WRITE REPORT TITLE
B4
WRITE FIRST COLUMN HEADINGS
C4
WRITE SECOND COLUMN HEADINGS
D4
SET SPACE CONTROL TO DOUBLE SPACE
Return to Caller
E4
F4
G4
H4
J4
K4
From F2
A5
300-END-SALES-CARDS
A5
CLEAR PRINTER AREA
B5
MOVE MESSAGE TO PRINT AREA
C5
MOVE TOTAL TO PRINT AREA
D5
WRITE FINAL TOTAL LINE
E5
CLOSE FILES
F5
STOP RUN
From C1
G5
600-SORT-IN
G5
OPEN UNSORTED FILE
H5
From K5
602-READ-UNSORTED
H5
READ UNSORTED CARDS
J5
MOVE TO SORT WORK AREA
K5
RELEASE TO SORT WORK FILE
H5

Figure 12-31 Flowchart

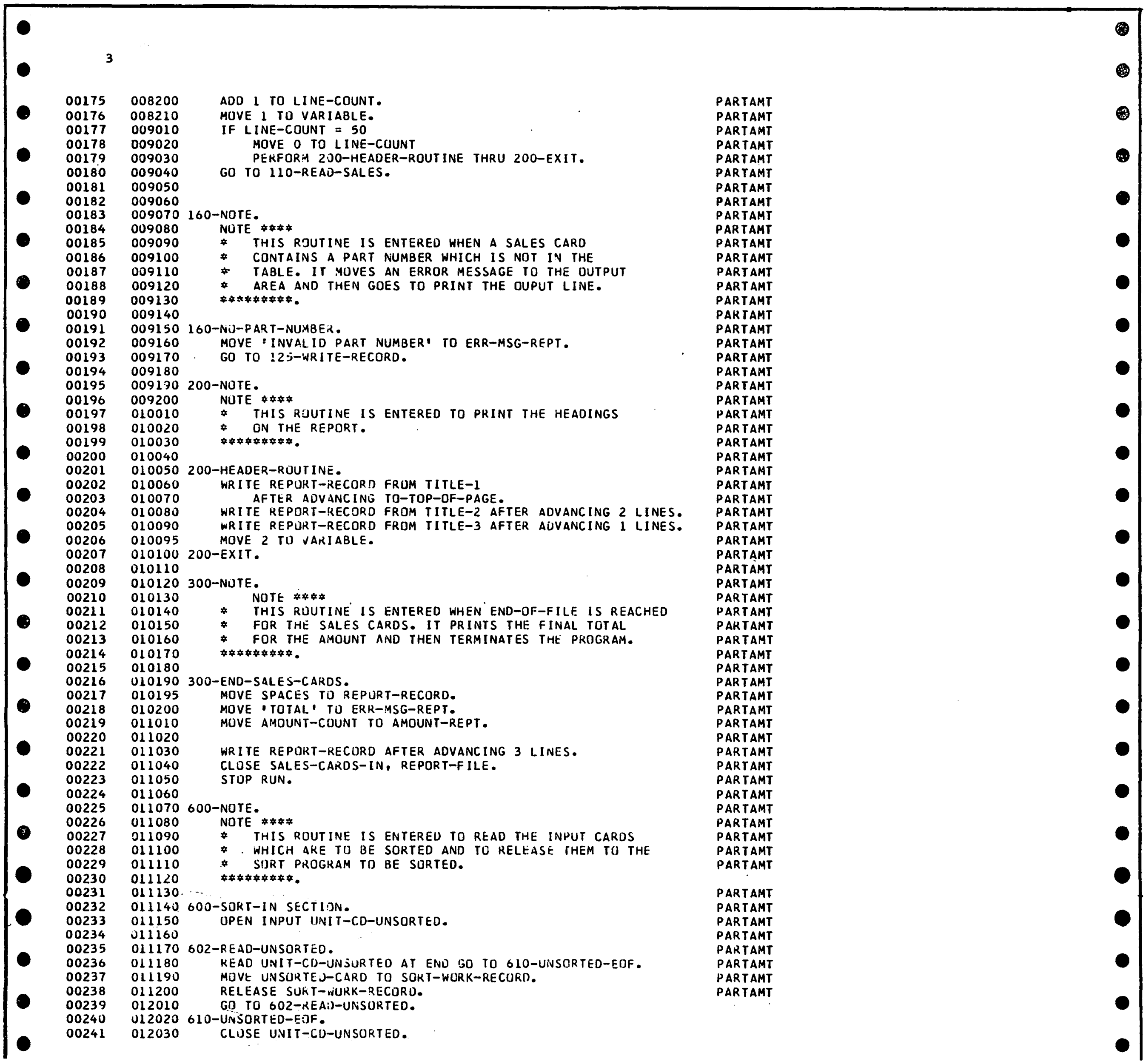


```
     1                                   IBM DOS AMERICAN NATIONAL STANDARD COBOL         CBF CL3-4             03/26/73

00001   001010 IDENTIFICATION DIVISION.                                         PARTAMT
00002   001020 PROGRAM-ID. PARTRPT.                                             PARTAMT
00003   001030 AUTHOR. TJC.                                                     PARTAMT
00004   001040 INSTALLATION. LONG BEACH.                                        PARTAMT
00005   001050 DATE-WRITTEN. 01/08/73.                                          PARTAMT
00006   001060 DATE-COMPILED. 03/26/73                                          PARTAMT
00007   001070 SECURITY. UNCLASSIFIED.                                          PARTAMT
00008   001080 REMARKS. THIS PROGRAM READS THE UNIT PRICE FILE OF CARDS         PARTAMT
00009   001090         WHICH ARE NORMALLY SORTED IN DEPARTMENT SEQUENCE         PARTAMT
00010   001100         AND RESORTS THEM INTO ASCENDING SEQUENCE BY PART         PARTAMT
00011   001110         NUMBER. THE RESORTED DATA IS THEN READ TO LOAD A         PARTAMT
00012   001120         TABLE IN SEQUENCE BY PART NUMBER. THE INPUT DATA         PARTAMT
00013   001130         CONTAINING THE PART NUMBER AND QUANTITY ARE THEN         PARTAMT
00014   001140         PROCESSED AGAINST THE TABLE AND THE REPORT IS            PARTAMT
00015   001150         CREATED.                                                 PARTAMT
00016   001160 ENVIRONMENT DIVISION.                                            PARTAMT
00017   001170 CONFIGURATION SECTION.                                           PARTAMT
00018   001180 SOURCE-COMPUTER. IBM-360-F30.                                    PARTAMT
00019   001190 OBJECT-COMPUTER. IBM-360-F30.                                    PARTAMT
00020   001192 SPECIAL-NAMES.                                                   PARTAMT
00021   001194     C01 IS TO-TOP-OF-PAGE.                                       PARTAMT
00022   001200 INPUT-OUTPUT SECTION.                                            PARTAMT
00023   002010 FILE-CONTROL.                                                    PARTAMT
00024   002020     SELECT UNIT-CD-UNSORTED                                      PARTAMT
00025   002030         ASSIGN TO SYS007-UR-2540R-S.                             PARTAMT
00026   002040     SELECT UNIT-CD-SORTED                                        PARTAMT
00027   002050         ASSIGN TO SYS016-UT-2314-S-SORTED.                       PARTAMT
00028   002060     SELECT SORT-WORK-FILE                                        PARTAMT
00029   002070         ASSIGN TO SYS001-UT-2314-S-SORTWK1.                      PARTAMT
00030   002080     SELECT REPORT-FILE                                           PARTAMT
00031   002090         ASSIGN TO SYS008-UR-1403-S.                              PARTAMT
00032   002092     SELECT SALES-CARDS-IN                                        PARTAMT
00033   002094         ASSIGN TO SYS009-UR-2540R-S.                             PARTAMT
00034   002100 DATA DIVISION.                                                   PARTAMT
00035   002110 FILE SECTION.                                                    PARTAMT
00036   002120 FD  UNIT-CD-UNSORTED                                             PARTAMT
00037   002130     RECORD CONTAINS 80 CHARACTERS,                               PARTAMT
00038   002140     LABEL RECORDS ARE OMITTED,                                   PARTAMT
00039   002150     DATA RECORD IS UNSORTED-CARD.                                PARTAM3
00040   002160 01  UNSORTED-CARD               PIC X(80).                       PARTAMT
00041   002170 FD  UNIT-CD-SORTED                                               PARTAMT
00042   002180     BLOCK CONTAINS 5 RECORDS                                     PARTAMT
00043   002190     RECORD CONTAINS 80 CHARACTERS                                PARTAMT
00044   002200     LABEL RECORDS ARE STANDARD                                   PARTAMT
00045   003010     DATA RECORD IS SORTED-CARD.                                  PARTAMT
00046   003020 01  SORTED-CARD.                                                 PARTAMT
00047   003030     03  DEPT-SC                 PIC XX.                          PARTAMT
00048   003040     03  PTNO-SC                 PIC XXXX.                        PARTAMT
00049   003050     03  UNIT-PRICE-SC           PIC 99V99.                       PARTAMT
00050   003060     03  FILLER                  PIC X(70).                       PARTAMT
00051   003070 SD  SORT-WORK-FILE                                               PARTAMT
00052   003090     RECORD CONTAINS 80 CHARACTERS                                PARTAMT
00053   003100     LABEL RECORDS ARE STANDARD                                   PARTAMT
00054   003110     DATA RECORD IS SORT-WORK-RECORD.                             PARTAMT
00055   003120 01  SORT-WORK-RECORD.                                            PARTAMT
00056   003130     03  DEPT-SW                 PIC XX.                          PARTAMT
00057   003140     03  PTNO-SW                 PIC XXXX.                        PARTAMT
00058   003150     03  UNIT-PRICE-SW           PIC 99V99.                       PARTAMT
00059   003160     03  FILLER                  PIC X(70).                       PARTAMT
00060   003170 FD  REPORT-FILE                                                  PARTAMT
00061   003171     RECORD CONTAINS 133 CHARACTERS                               PARTAMT
00062   003172     LABEL RECORDS ARE OMITTED                                    PARTAMT
00063   003173     DATA RECORD IS REPORT-RECORD.                                PARTAMT
00064   003179 01  REPORT-RECORD.                                               PARTAMT
00065   003180     03  FILLER                  PIC XXXX.                        PARTAMT
00066   003190     03  PTNO-REPT               PIC XXXX.                        PARTAMT
00067   003200     03  FILLER                  PIC X(8).                        PARTAMT
00068   004010     03  DEPT-REPT               PIC XX.                          PARTAMT
00069   004020     03  FILLER                  PIC X(8).                        PARTAMT
00070   004030     03  QTY-REPT                PIC ZZ9.                         PARTAMT
00071   004040     03  FILLER                  PIC X(5).                        PARTAMT
00072   004045     02  ERR-MSG-REPT.                                            PARTAMT
00073   004050     03  UNIT-PRICE-REPT         PIC $$$.99.                      PARTAMT
00074   004060     03  FILLER                  PIC XXXX.                        PARTAMT
00075   004070     03  AMOUNT-REPT             PIC $$$,$$$.99.                  PARTAMT
00076   004080     03  FILLER                  PIC X(79).                       PARTAMT
00077   004090 FD  SALES-CARDS-IN                                               PARTAMT
00078   004100     RECORD CONTAINS 80 CHARACTERS                                PARTAMT
00079   004110     LABEL RECORDS ARE OMITTED                                    PARTAMT
00080   004120     DATA RECORD IS SALES-CARD.                                   PARTAMT
00081   004130 01  SALES-CARD.                                                  PARTAMT
00082   004140     03  PTNO-SAL                PIC XXXX.                        PARTAMT
00083   004150     03  QTY-SAL                 PIC 999.                         PARTAMT
00084   004160     03  FILLER                  PIC X(73).                       PARTAMT
00085   004170 WORKING-STORAGE SECTION.                                         PARTAMT
00086   004180                                                                  PARTAMT
00087   004198 77  VARIABLE                    PIC 9.                           PARTAMT
```

Figure 12-32 Source Program (Part 1 of 3)

```
     2

00088   004190 77  AMOUNT-COUNT                PIC S9(5)V99   VALUE ZERO COMP-3.PARTAMT
00089   004195 77  TEMP-AMOUNT                 PIC S9(5)V99    COMP-3 VALUE  0. PARTAMT
00090   004197 77  LINE-COUNT                  PIC S999        COMP-3 VALUE 0.  PARTAMT
00091   004200 01  ALIGNMENT-01.                                                PARTAMT
00092   005010     03  TITLE-1.                                                 PARTAMT
00093   005020         05  FILLER              PIC X(13)       VALUE SPACES.    PARTAMT
00094   005030         05  FILLER              PIC X(120)      VALUE            PARTAMT
00095   005040             'S A L E S    R E P O R T'.                          PARTAMT
00096   005050     03  TITLE-2.                                                 PARTAMT
00097   005060         05  FILLER              PIC X(4)        VALUE SPACES.    PARTAMT
00098   005070         05  FILLER              PIC X(32)       VALUE 'PART'.    PARTAMT
00099   005080         05  FILLER              PIC X(97)       VALUE 'UNIT'.    PARTAMT
00100   005090     03  TITLE-3.                                                 PARTAMT
00101   005100         05  FILLER              PIC XXX         VALUE SPACES.    PARTAMT
00102   005110         05  FILLER              PIC X(12)       VALUE 'NUMBER'.  PARTAMT
00103   005120         05  FILLER              PIC X(9)        VALUE 'DEPT'.    PARTAMT
00104   005130         05  FILLER              PIC X(11)       VALUE 'QUANTITY'.PARTAMT
00105   005140         05  FILLER              PIC X(11)       VALUE 'PRICE'.   PARTAMT
00106   005150         05  FILLER              PIC X(87)       VALUE 'AMOUNT'.  PARTAMT
00107   005160     03  UNIT-PRICE-TBL     OCCURS 20 TIMES                       PARTAMT
00108   005170             ASCENDING KEY IS PTNO-TABLE                          PARTAMT
00109   005180             INDEXED BY UP-INDEX.
00110   005185         05  DEPT-TABLE          PIC XX.                          PARTAMT
00111   005190         05  PTNO-TABLE          PIC XXXX.                        PARTAMT
00112   005200         05  UNIT-PRICE-TABLE    PIC 99V99.                       PARTAMT
00113   006010 PROCEDURE DIVISION.                                              PARTAMT
00114   006020 001-FIRST-NOTE.                                                  PARTAMT
00115   006030     NOTE ****                                                    PARTAMT
00116   006040     *   THE FIRST PORTION OF THE PROCEDURE DIVISION DEALS WITH   PARTAMT
00117   006050     *   READING IN THE UNIT PRICE CARDS AND SORTING THEM ON      PARTAMT
00118   006060     *   THE PART NUMBER. AFTER THEY ARE SORTED ON PART NUMBER,   PARTAMT
00119   006070     *   THE SORTED DATA WILL BE READ BACK IN AND PLACED IN       PARTAMT
00120   006080     *   A TABLE. THE SALES INPUT CARDS WILL THEN BE READ AND     PARTAMT
00121   006090     *   THE TABLE IS SEARCHED FOR THE CORRECT UNIT PRICE.        PARTAMT
00122   006100     *   WHEN FOUND, THE UNIT PRICE IS MULTIPLIED BY THE          PARTAMT
00123   006110     *   QTY TO GET THE AMOUNT AND THEN THE REPORT IS WRITTEN.    PARTAMT
00124   006120     *********.                                                   PARTAMT
00125   006130 001-SORT-CARDS.                                                  PARTAMT
00126   006140     SORT  SORT-WORK-FILE                                         PARTAMT
00127   006150           ASCENDING KEY PTNO-SW                                  PARTAMT
00128   006160           INPUT PROCEDURE IS 600-SORT-IN                         PARTAMT
00129   006170           GIVING UNIT-CD-SORTED.                                 PARTAMT
00130   006180 020-BUILD-TABLE SECTION.                                         PARTAMT
00131   006190     OPEN INPUT UNIT-CD-SORTED.                                   PARTAMT
00132   006200                                                                  PARTAMT
00133   007010     SET UP-INDEX TO 1.                                           PARTAMT
00134   007020 023-READ-SORTED-INPUT.                                           PARTAMT
00135   007030     READ UNIT-CD-SORTED AT END GO TO 029-TABLE-BUILT.            PARTAMT
00136   007040     MOVE PTNO-SC TO PTNO-TABLE (UP-INDEX).                       PARTAMT
00137   007060     MOVE UNIT-PRICE-SC TO UNIT-PRICE-TABLE (UP-INDEX).           PARTAMT
00138   007065     MOVE DEPT-SC TO DEPT-TABLE (UP-INDEX).                       PARTAMT
00139   007070     SET UP-INDEX UP BY 1.                                        PARTAMT
00140   007080     GO TO 023-READ-SORTED-INPUT.                                 PARTAMT
00141   007090 029-TABLE-BUILT.                                                 PARTAMT
00142   007100     CLOSE UNIT-CD-SORTED.                                        P1RT1MT
00143   007110                                                                  PARTAMT
00144   007120 100-WRITE-REPORT-NOTE.                                           PARTAMT
00145   007130     NOTE ****                                                    PARTAMT
00146   007140     *   THIS ROUTINE IS ENTERED AFTER THE UNIT PRICE CARDS       PARTAMT
00147   007150     *   HAVE BEEN SORTED AND THEN READ BACK INTO THE UNIT        PARTAMT
00148   007160     *   PRICE TABLE. THE DETAIL SALES CARDS ARE READ, THE        PARTAMT
00149   007170     *   UNIT PRICE IS FOUND IN THE TABLE, AND THE REPORT         PARTAMT
00150   007180     *   IS WRITTEN.                                              PARTAMT
00151   007190     *********.                                                   PARTAMT
00152   007200                                                                  PARTAMT
00153   008010 100-WRITE-REPORT.                                                PARTAMT
00154   008020     OPEN INPUT SALES-CARDS-IN, OUTPUT REPORT-FILE.               PARTAMT
00155   008025     PERFORM 200-HEADER-ROUTINE THRU 200-EXIT.                    PARTAMT
00156   008030     SET UP-INDEX TO 1.                                           PARTAMT
00157   008040 110-READ-SALES.                                                  PARTAMT
00158   008050     READ SALES-CARDS-IN AT END GO TO 300-END-SALES-CARDS.        PARTAMT
00159   008055     MOVE  SPACES TO REPORT-RECORD.                               PARTAMT
00160   008060     MOVE PTNO-SAL TO PTNO-REPT.                                  PARTAMT
00161   008070     MOVE QTY-SAL TO QTY-REPT.                                    PARTAMT
00162   008080     SEARCH ALL UNIT-PRICE-TBL                                    PARTAMT
00163   008090             AT END GO TO 160-NO-PART-NUMBER                      PARTAMT
00164   008100             WHEN PTNO-SAL = PTNO-TABLE (UP-INDEX)                PARTAMT
00165   008110                 GO TO 120-PTNO-FOUND.                            PARTAMT
00166   008120 120-PTNO-FOUND.                                                  PARTAMT
00167   008130     MULTIPLY UNIT-PRICE-TABLE (UP-INDEX) BY QTY-SAL              PARTAMT
00168   008140         GIVING TEMP-AMOUNT.                                      PARTAMT
00169   008150     ADD TEMP-AMOUNT TO AMOUNT-COUNT.                             PARTAMT
00170   008160     MOVE TEMP-AMOUNT TO AMOUNT-REPT.                             PARTAMT
00171   008170     MOVE DEPT-TABLE (UP-INDEX) TO DEPT-REPT.                     PARTAMT
00172   008180     MOVE UNIT-PRICE-TABLE (UP-INDEX) TO UNIT-PRICE-REPT.         PARTAMT
00173   008189 125-WRITE-RECORD.                                                PARTAMT
00174   008190     WRITE REPORT-RECORD AFTER ADVANCING VARIABLE LINES.          PARTAMT
```

Figure 12-33 Source Program (Part 2 of 3)



```
       3

  00175    008200      ADD 1 TO LINE-COUNT.                                      PARTAMT
  00176    008210      MOVE 1 TO VARIABLE.                                       PARTAMT
  00177    009010      IF LINE-COUNT = 50                                        PARTAMT
  00178    009020          MOVE 0 TO LINE-COUNT                                  PARTAMT
  00179    009030          PERFORM 200-HEADER-ROUTINE THRU 200-EXIT.             PARTAMT
  00180    009040      GO TO 110-READ-SALES.                                     PARTAMT
  00181    009050                                                                PARTAMT
  00182    009060                                                                PARTAMT
  00183    009070 160-NOTE.                                                      PARTAMT
  00184    009080      NOTE ****                                                 PARTAMT
  00185    009090      *   THIS ROUTINE IS ENTERED WHEN A SALES CARD             PARTAMT
  00186    009100      *   CONTAINS A PART NUMBER WHICH IS NOT IN THE            PARTAMT
  00187    009110      *   TABLE. IT MOVES AN ERROR MESSAGE TO THE OUTPUT        PARTAMT
  00188    009120      *   AREA AND THEN GOES TO PRINT THE OUPUT LINE.           PARTAMT
  00189    009130      *********.                                                PARTAMT
  00190    009140                                                                PARTAMT
  00191    009150 160-NO-PART-NUMBER.                                            PARTAMT
  00192    009160      MOVE 'INVALID PART NUMBER' TO ERR-MSG-REPT.               PARTAMT
  00193    009170      GO TO 125-WRITE-RECORD.                                   PARTAMT
  00194    009180                                                                PARTAMT
  00195    009190 200-NOTE.                                                      PARTAMT
  00196    009200      NOTE ****                                                 PARTAMT
  00197    010010      *   THIS ROUTINE IS ENTERED TO PRINT THE HEADINGS         PARTAMT
  00198    010020      *   ON THE REPORT.                                        PARTAMT
  00199    010030      *********.                                                PARTAMT
  00200    010040                                                                PARTAMT
  00201    010050 200-HEADER-ROUTINE.                                            PARTAMT
  00202    010060      WRITE REPORT-RECORD FROM TITLE-1                          PARTAMT
  00203    010070          AFTER ADVANCING TO-TOP-OF-PAGE.                       PARTAMT
  00204    010080      WRITE REPORT-RECORD FROM TITLE-2 AFTER ADVANCING 2 LINES. PARTAMT
  00205    010090      WRITE REPORT-RECORD FROM TITLE-3 AFTER ADVANCING 1 LINES. PARTAMT
  00206    010095      MOVE 2 TO VARIABLE.                                       PARTAMT
  00207    010100 200-EXIT.                                                      PARTAMT
  00208    010110                                                                PARTAMT
  00209    010120 300-NOTE.                                                      PARTAMT
  00210    010130          NOTE ****                                             PARTAMT
  00211    010140      *   THIS ROUTINE IS ENTERED WHEN END-OF-FILE IS REACHED   PARTAMT
  00212    010150      *   FOR THE SALES CARDS. IT PRINTS THE FINAL TOTAL        PARTAMT
  00213    010160      *   FOR THE AMOUNT AND THEN TERMINATES THE PROGRAM.       PARTAMT
  00214    010170      *********.                                                PARTAMT
  00215    010180                                                                PARTAMT
  00216    010190 300-END-SALES-CARDS.                                           PARTAMT
  00217    010195      MOVE SPACES TO REPORT-RECORD.                             PARTAMT
  00218    010200      MOVE 'TOTAL' TO ERR-MSG-REPT.                             PARTAMT
  00219    011010      MOVE AMOUNT-COUNT TO AMOUNT-REPT.                         PARTAMT
  00220    011020                                                                PARTAMT
  00221    011030      WRITE REPORT-RECORD AFTER ADVANCING 3 LINES.              PARTAMT
  00222    011040      CLOSE SALES-CARDS-IN, REPORT-FILE.                        PARTAMT
  00223    011050      STOP RUN.                                                 PARTAMT
  00224    011060                                                                PARTAMT
  00225    011070 600-NOTE.                                                      PARTAMT
  00226    011080      NOTE ****                                                 PARTAMT
  00227    011090      *   THIS ROUTINE IS ENTERED TO READ THE INPUT CARDS       PARTAMT
  00228    011100      *   WHICH ARE TO BE SORTED AND TO RELEASE THEM TO THE     PARTAMT
  00229    011110      *   SORT PROGRAM TO BE SORTED.                            PARTAMT
  00230    011120      *********.
  00231    011130                                                                PARTAMT
  00232    011140 600-SORT-IN SECTION.                                           PARTAMT
  00233    011150      OPEN INPUT UNIT-CD-UNSORTED.                              PARTAMT
  00234    011160                                                                PARTAMT
  00235    011170 602-READ-UNSORTED.                                             PARTAMT
  00236    011180      READ UNIT-CD-UNSORTED AT END GO TO 610-UNSORTED-EOF.      PARTAMT
  00237    011190      MOVE UNSORTED-CARD TO SORT-WORK-RECORD.                   PARTAMT
  00238    011200      RELEASE SORT-WORK-RECORD.                                 PARTAMT
  00239    012010      GO TO 602-READ-UNSORTED.
  00240    012020 610-UNSORTED-EOF.
  00241    012030      CLOSE UNIT-CD-UNSORTED.
```

Figure 12-34 Source Program (Part 3 of 3)

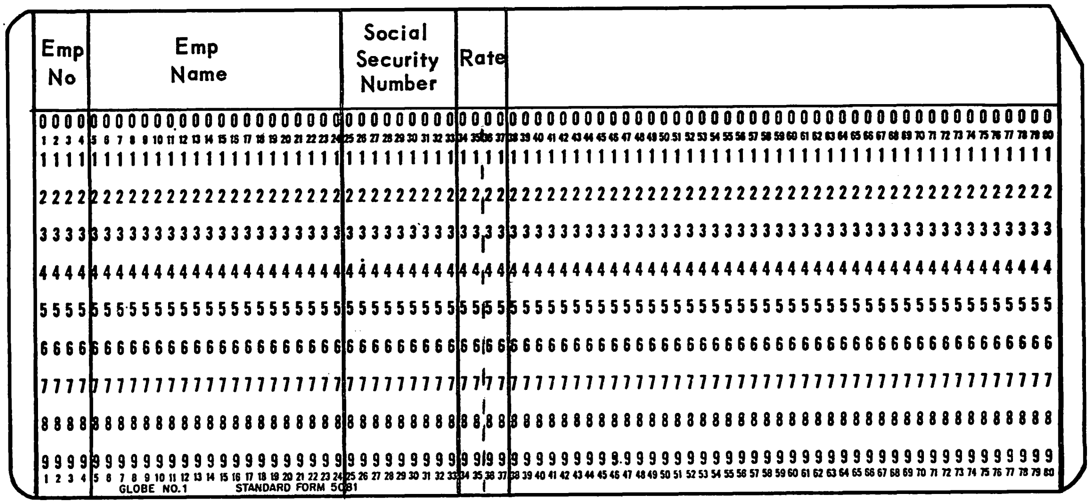
PROGRAMMING ASSIGNMENT 1

INSTRUCTIONS

Two reports are to be produced from the file of cards illustrated below. One report must be in employee number sequence and the other report must be in social security number sequence. The file of cards is normally kept in employee number sequence and this is the sequence in which the cards are to be read by the program. Write a single program to produce these reports.

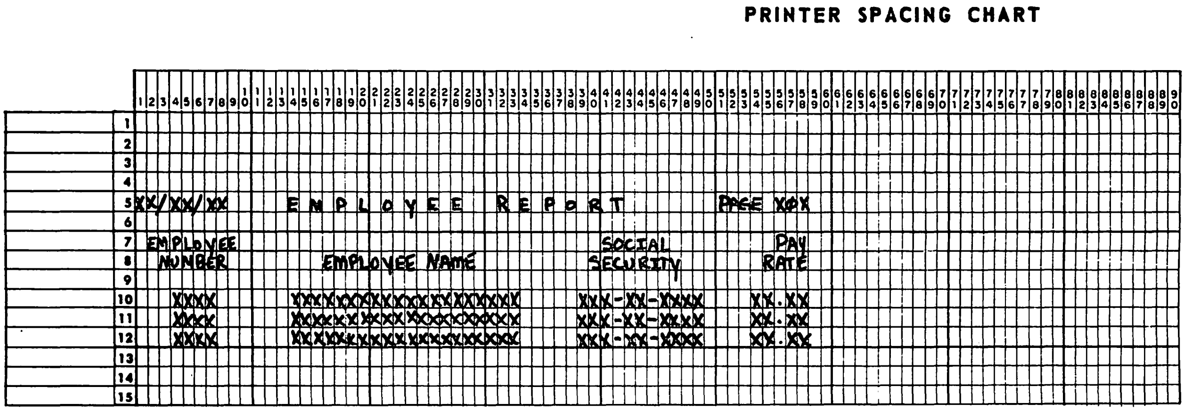
INPUT

The format of the Employee Cards is illustrated below.


Emp No
Emp Name
Social Security Number
Rate
GLOBE NO. 1
STANDARD FORM 5081

OUTPUT

The output is to be two reports – one in employee number sequence and one in social security number sequence. The two reports are to use the same format, as illustrated below.


PRINTER SPACING CHART
XX/XX/XX
EMPLOYEE REPORT
PAGE XØX
EMPLOYEE NUMBER
EMPLOYEE NAME
SOCIAL SECURITY
PAY RATE
XXXX
XXXXXXXXXXXXXXXXXXXXXX
XXX-XX-XXXX
XX.XX

TEST DATA – ASSIGNMENT NO. 1

EMP NO (col 1-4)	EMP NAME (col 5-24)	SOCIAL SECURITY NUMBER (col 25-33)	PAY RATE (col 34-37)
0274	ALFS, RUDOLPH	276 33 9911	0245
0983	BOWES, RALPH	926 99 3342	0850
1237	DOLDOO, JEREMIAH	073 44 9236	0250
4837	McBUN, NANCY	074 26 5501	0640
7243	SCHMALTZ, BERNIE	174 25 3389	0500
8432	TORNO, RANDY	755 21 0022	0980

CHAPTER 12

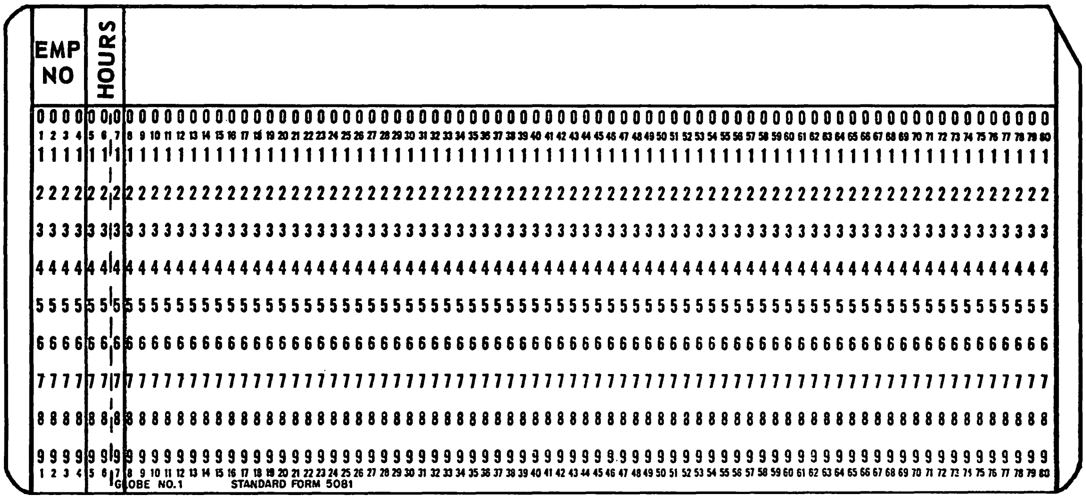
PROGRAMMING ASSIGNMENT 2

INSTRUCTIONS

The data cards illustrated in Programming Assignment 1 are to be used to build a table so that the pay rate, social security number, and employee name can be extracted and used in preparing a payroll report. The table is to be in employee number sequence as the input cards, as illustrated below, contain only the employee number. However, due to government regulations, the payroll report must be in social security number sequence. Write a single COBOL program to produce the payroll report.

INPUT

The format of the input cards for the employees to be contained on the report is illustrated below.



OUTPUT

The output is the Payroll Report which is to be produced. The programmer should design the format of the report which is to be in social-security number sequence and is to contain the Social Security Number, the Employee Number, the Employee Name, the Pay Rate, the Hours Worked, and the Gross Earnings. It should be noted that if a detail card is read for which there is not a matching employee number in the table created from the cards in Programming Assignment 1, the message "INVALID EMPLOYEE NUMBER" should be printed together with the employee number for identification purposes.

TEST DATA

Employee Number (col 1-4)	Hours (col 5-7)
0983	40.7
1237	52.3
6432	28.1
7243	40.0
8627	48.6

CHAPTER 13

ADDITIONAL COBOL STATEMENTS

The previous Chapters have introduced many of the basic programming concepts and COBOL statements necessary to solve a wide variety of business problems. A number of additional options and statements are also available. Some of these statements are summarized on the following pages.

ACCEPT Statement

The function of the ACCEPT statement is to obtain data from the system logical input device or from the console.

The format of the ACCEPT Statement is:

```
ACCEPT Identifier FROM mnemonic-name
```

Figure 13-1 Format Notation – ACCEPT Statement

"Identifier" may be either a fixed-length group item or an elementary alphabetic, alphanumeric, or external decimal item. One logical record is read and the appropriate number of characters are transferred from left to right into the area reserved for "identifier". No editing or error checking on the incoming data is performed. If the input/output device used with the ACCEPT statement is the same as the device designated for a READ statement, the results may be unpredictable.

"Mnemonic-name" will normally assume the meaning CONSOLE. Mnemonic-name must be specified in the Special-Names paragraph of the Environment Division in much the same manner as for the DISPLAY statement. If mnemonic-name is associated with CONSOLE, identifier must not exceed 255 character positions in length. The following is an example of the ACCEPT statement.

```
 SPECIAL-NAMES.
     CONSOLE IS TYPEWRITER.

     ACCEPT DATE FROM TYPEWRITER.
```

Figure 13-2 Example of the Accept Statement

PERFORM Statement

The PERFORM Statement specifies a transfer of control from one portion of a program to another, in order to execute some procedure a specified number of times, or until a condition is satisfied. It directs that control is to be returned to the statement immediately following the point from which the transfer was made.

The PERFORM Statement has the following format:

```
Option 1

PERFORM procedure-name-1 [THRU procedure-name-2]
```

Figure 13-3 Format Notation – Option 1 Perform Statement

Option 1 is the simple PERFORM Statement. A procedure referred to by this type of PERFORM Statement is executed once, and then control passes to the next statetment after the PERFORM Statement. All statements in the paragraphs or sections named by "Procedure-Name-1" through "Procedure-Name-2" constitute the range of the PERFORM Statement. The following is an illustration of the use of this statement.

EXAMPLE 1:

```
    PERFORM MINOR-ROUTINE.
```

EXAMPLE 2:

```
    PERFORM MINOR-ROUTINE THRU MAJOR-ROUTINE.
```

Figure 13-4 Example of the Perform Statement

Option 2 is the TIMES option. When the TIMES option is used, the procedure is performed the number of times specified by "identifier-1" or "Integer-1". Control is then transferred to the statement following the PERFORM statement. "Identifier-1" must have an integral value and "Identifier-1" or "Integer-1" must have a positive value. If the value of the "Identifier-1" is negative or zero, control is passed immediately to the statement following the PERFORM statement.

The following examples illustrate the format notation and examples of Option 2 of the Perform statement.

```
PERFORM procedure-name-1 [THRU procedure-name-2]
   { identifier-1 }
   {              }  TIMES
   { integer-1    }
```

EXAMPLE 1:

```
    PERFORM PRINT-ROUTINE 5 TIMES.
```

EXAMPLE 2:

```
    PERFORM PENALTY-ROUTINE MONTHS-LATE TIMES.
```

Figure 13-5 Example of Option 2 of the Perform Statement

Option 3 is the UNTIL Option. "Condition-1" may be simple or compound. The procedures specified by the procedure-names are performed until the condition specified by the UNTIL Option is true. At this time, control is transferred to the statement following the PERFORM Statement. If the condition specified by the UNTIL Option is true at the time the PERFORM Statement is encountered, the specified procedure is not executed.

The following examples illustrate the format notation and example of Option 3 of the Perform Statement.

```
PERFORM procedure-name-1 [THRU procedure-name-2]
    UNTIL condition-1
```

EXAMPLE 1:

```
PERFORM PROCESS-ROUTINE UNTIL CONTROL-IN IS EQUAL TO 25.
```

Figure 13-6 Example of Option 3 of the Perform Statement

Option 4 is the VARYING Option. The Condition specified may be simple or compound. The following example illustrates the use of this option.

```
PERFORM procedure-name-1 [THRU procedure-name-2]

          ( index-name-1 )       ( index-name-2 )
  VARYING <              >  FROM < literal-2    >
          ( identifier-1 )       ( identifier-2 )

      ( literal-3    )
  BY  <              >  UNTIL  condition-1
      ( identifier-3 )

          ( index-name-4 )       ( index-name-5 )
  [AFTER  <              >  FROM < literal-5    >
          ( identifier-4 )       ( identifier-5 )

      ( literal-6    )
  BY  <              >  UNTIL condition-2
      ( identifier-6 )

          ( index-name-7 )       ( index-name-8 )
  [AFTER  <              >  FROM < literal-8    >
          ( identifier-7 )       ( identifier-8 )

      ( literal-9    )
  BY  <              >  UNTIL condition-3]]
      ( identifier-9 )
```

EXAMPLE 1:

```
PERFORM PROFIT-SHARING-ROUTINE VARYING AGE FROM HIRED-DATE
    BY 1 UNTIL AGE IS EQUAL TO 65.
```

EXAMPLE 2:

```
PERFORM READ-ROUTINE VARYING COUNT FROM 1 BY 1,
    UNTIL COUNT IS EQUAL TO 10.
```

Figure 13-7 Example of Option 4 of the Perform Statement

EXAMINE Statement

One example of the EXAMINE Statement was illustrated in Chapter 7. A more detailed explanation is presented below.

The EXAMINE Statement is used to replace certain occurrences of a given character and/or to count the number of such occurrences in a data item.

The EXAMINE Statement has the following two formats:

```
Format 1
                                   ( UNTIL FIRST )
  EXAMINE identifier TALLYING     {  ALL          }  literal-1
                                   ( LEADING     )

          [REPLACING BY literal-2]


Format 2
                                   ( ALL         )
                                   ) LEADING     (
  EXAMINE identifier REPLACING    {  FIRST        }  literal-1
                                   ( UNTIL FIRST )

          BY  literal-2
```

Figure 13-8 Format Notation – Examine Statement

"Identifier" in each option must refer to a data item whose USAGE is DISPLAY.

"Literal-1" and "Literal-2" must be single-character non-numeric literals (i.e., enclosed in quotation marks) and members of the set of allowable characters for the data item. For example, a "2" cannot replace an "A" in an alphabetic item, but may do so in an alphanumeric item.

The use of figurative constants except ALL instead of "Literal-1" or "Literal-2" is permitted.

When Format 1 is used, a count is made at object time of the number of occurrences of the specified character in "Identifier", and this count replaces the value of the special binary data item TALLY, whose length is five decimal digits. Tally may also be used as a data-name in other procedural statements.

The count at object time depends on which of the following three TALLYING options is employed:

1. If ALL is specified, all occurrences of "Literal-1" in the data item are counted.
2. If LEADING is specified, the count represents the number of occurrences of "Literal-1" prior to encountering a character other than "Literal-1". Examination proceeds from left to right.
3. If UNTIL FIRST is specified, the count represents the number of characters other than "Literal-1" encountered prior to the first occurrence of "Literal-1". Examination proceeds from left to right.

EXAMPLE 1:

```
    EXAMINE ITEM-1 TALLYING ALL '0'.
```

EXAMPLE 2:

```
    EXAMINE ITEM-1 TALLYING ALL '1' REPLACING BY '0'.
```

Figure 13-9 Example of Examine Statement – Tallying Option

The following chart illustrates the value of TALLY after execution of the EXAMINE Statement illustrated in Examples 1 and Examples 2 above.

EXAMINE Statement	ITEM-1 Before	Data After	Resulting Value of TALLY
EXAMINE ITEM-1 TALLYING ALL '0'	101010	101010	3
EXAMINE ITEM-1 TALLYING ALL '1' REPLACING BY '0'	101010	000000	3

When the REPLACING option is used (either in format 1 or format 2), the replacement of characters depends on which of the following four REPLACING options is employed:

1. If ALL is specified, "Literal-2" is substituted for each occurrence of "Literal-1".
2. If LEADING is specified, the substitution of "Literal-2" for "Literal-1" terminates when a character other then "Literal-1" is encountered, or when the righthand boundary of the data item is reached. Examination proceeds from left to right.
3. If UNTIL FIRST is specified, the substitution of "Literal-2" terminates as soon as the first "Literal-1" is encountered, or when the righthand boundary is reached. Examination proceeds from left to right.
4. If FIRST is specified, only the first occurrence of "Literal-1" is replaced by "Literal-2". Examination proceeds from left to right.

EXAMPLE 1

```
EXAMINE ITEM-1 REPLACING LEADING '*' BY SPACE.
```

EXAMPLE 2:

```
EXAMINE ITEM-1 REPLACING FIRST '*' BY '#'.
```

Figure 13-10 Example of Examine Statement – Replacing Option

The following chart illustrates the contents of ITEM-1 after execution of the EXAMINE Statement illustrated in Examples 1 and Examples 2.

EXAMINE Statement	ITEM-1 Before	Data After	Resulting Value of TALLY
EXAMINE ITEM-1 REPLACING LEADING '*' BY SPACE	**7000	7000	unchanged
EXAMINE ITEM-1 REPLACING FIRST '*' BY '$'	**1.94	$*1.94	unchanged

ALTER Statement

The ALTER Statement is used to modify an unconditional GO TO Statement elsewhere in the Procedure Division, thus changing the sequence in which program steps are to be executed.

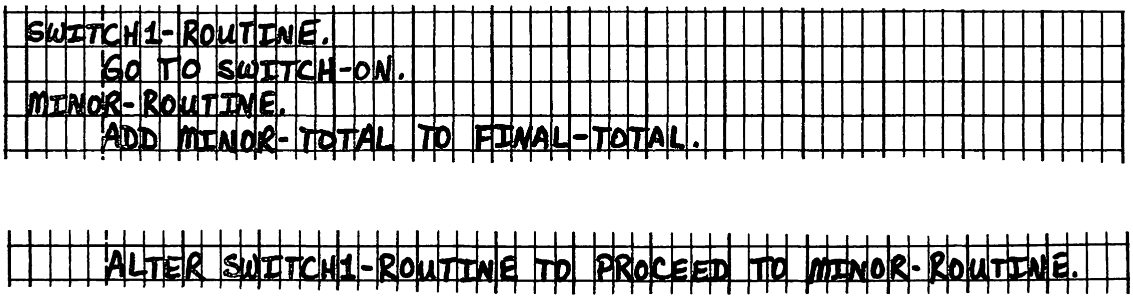
The format of the ALTER Statement is:

```
ALTER  procedure-name-1 TO [PROCEED TO] procedure-name-2

       [procedure-name-3 TO [PROCEED TO] procedure-name-4] . . .
```

Figure 13-11 Format Notation – Alter Statement

"Procedure-Name-1" designates a paragraph containing a single sentence consisting only of a GO TO statement. The effect of an ALTER statement is to replace the procedure-name specified in the GO TO statement with "Procedure-Name-2" of the ALTER statement, where the paragraph-name containing the GO TO statement is "Procedure-Name-1" in the ALTER statement.

The Alter Statement is frequently used in programs requiring a programmed switch. The following example illustrates a segment of the Procedure Division using the Alter Statement as part of the programmed switch.



```
SWITCH1-ROUTINE.
    GO TO SWITCH-ON.
MINOR-ROUTINE.
    ADD MINOR-TOTAL TO FINAL-TOTAL.

    ALTER SWITCH1-ROUTINE TO PROCEED TO MINOR-ROUTINE.
```

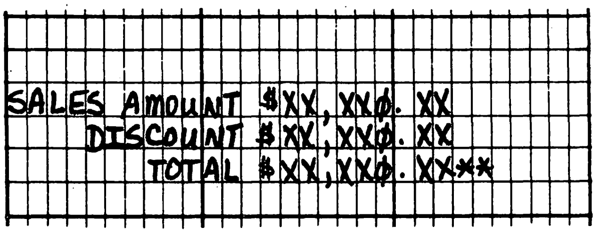
Figure 13-12 Example of Alter Statement

When the Alter Statement is executed the statement GO TO SWITCH-ON is effectively changed to read GO TO MINOR-ROUTINE.

JUSTIFIED RIGHT Clause

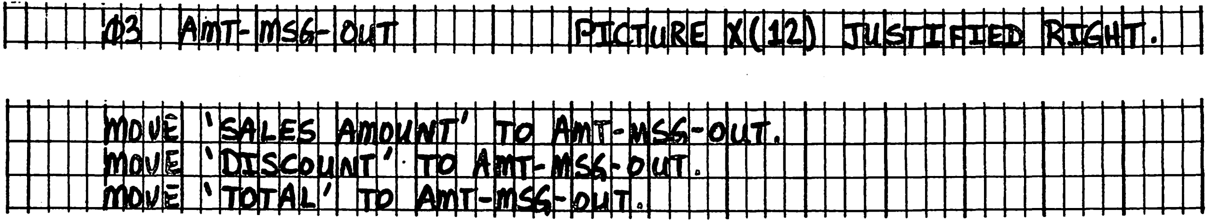
In some applications it may be desirable to have alphabetic data right justified. Normally an alphabetic move is left justified. For example in the following segment of the printer spacing chart illustrated below the words DISCOUNT and TOTAL are right justified. This may be accomplished by use of the JUSTIFIED RIGHT Clause in the Data Division. The following segment of a program illustrates the use of the justified Right Clause.

PRINTER SPACING CHART



```
SALES AMOUNT $XX,XX0.XX
    DISCOUNT $XX,XX0.XX
       TOTAL $XX,XX0.XX**
```

CODING



```
03  AMT-MSG-OUT          PICTURE X(12) JUSTIFIED RIGHT.

    MOVE 'SALES AMOUNT' TO AMT-MSG-OUT.
    MOVE 'DISCOUNT' TO AMT-MSG-OUT.
    MOVE 'TOTAL' TO AMT-MSG-OUT.
```

Figure 13-13 Example of Justified Right Clause

The Justified Right Clause may be written only for an elementary alphabetic or alphanumeric item. Its format is

```
[JUSTIFIED RIGHT]
```

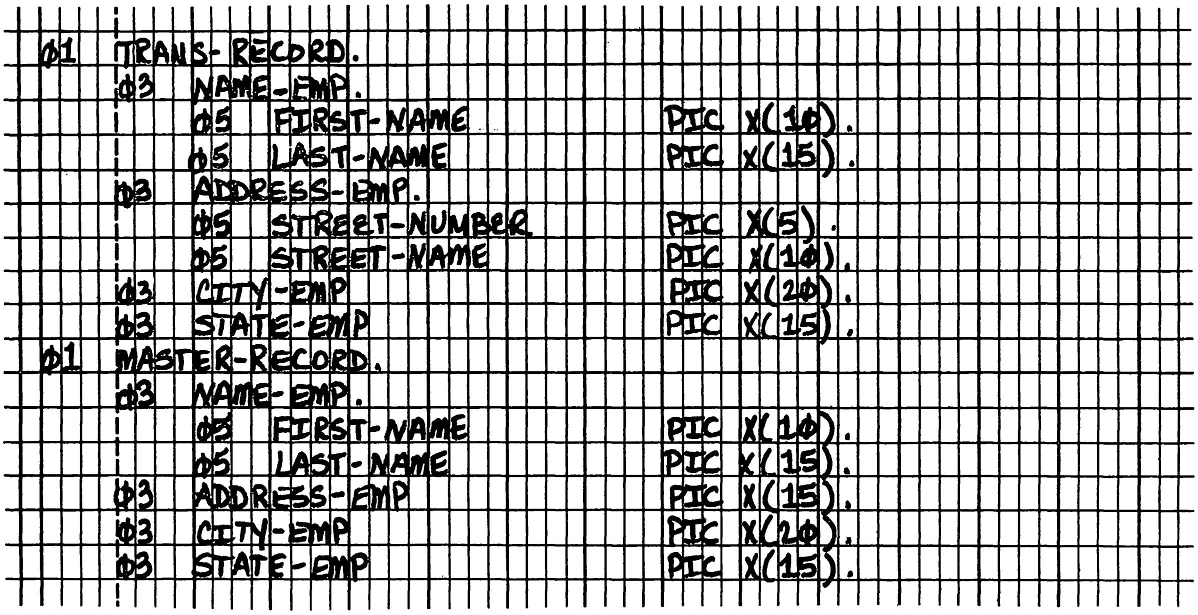
Figure 13-14 Format Notation – Justified Right Clause

When non-numeric data is moved to a field for which JUSTIFIED RIGHT has been specified, the rightmost character of the source field is placed in the rightmost position of the receiving field. The moving of characters continues from right to left until the receiving field is filled. If the length of the source field is greater than that of the receiving field, truncation terminates the move after the leftmost position of the receiving field is filled. If the source field is shorter, the remaining leftmost positions of the receiving field are filled with spaces.

NAME QUALIFICATION

In previous examples of defining data within either the File Section of the Data Division or the Working-Storage Section, all data-names and identifiers which have been used have been unique, that is, each has been different. It is allowable, however, to have the same data-names reference different areas of main storage. The requirement, however, is that there be some higher-level name which is unique and which will Qualify the data-name and therefore make it unique.

The ability to qualify names is illustrated in Figure 13-15.



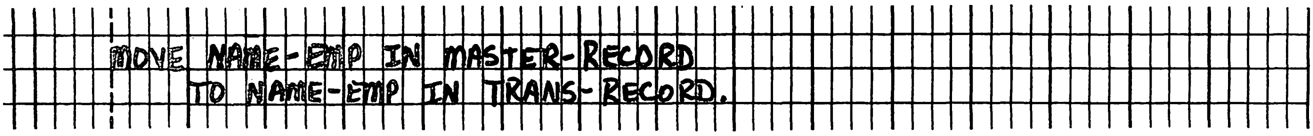
```
01  TRANS-RECORD.
    03  NAME-EMP.
        05  FIRST-NAME          PIC X(10).
        05  LAST-NAME           PIC X(15).
    03  ADDRESS-EMP.
        05  STREET-NUMBER       PIC X(5).
        05  STREET-NAME         PIC X(10).
    03  CITY-EMP                PIC X(20).
    03  STATE-EMP               PIC X(15).
01  MASTER-RECORD.
    03  NAME-EMP.
        05  FIRST-NAME          PIC X(10).
        05  LAST-NAME           PIC X(15).
    03  ADDRESS-EMP             PIC X(15).
    03  CITY-EMP                PIC X(20).
    03  STATE-EMP               PIC X(15).
```

Figure 13-15 Example of Name-Qualification

In the example in Figure 13-15 it can be seen that there are two 01 level data-names, TRANS-RECORD and MASTER-RECORD. Within these record descriptions there are both group and elementary items with identical names, that is, both records have a NAME-EMP group field, a FIRST-NAME elementary item, etc. It will be recalled that in all previous programs the data-names have been unique. In order to reference these different fields with the same names, each of the identical names must be "qualified" up to a point where they become unique. Thus, if one were to reference the name NAME-EMP as belonging to the group TRANS-RECORD or belonging to the group MASTER-RECORD, then these data-names would become unique. This process is known as qualification.

The example in Figure 13-16 illustrates a MOVE statement which could be used to move the NAME-EMP field in the MASTER-RECORD to the NAME-EMP field in the TRANS-RECORD.

EXAMPLE



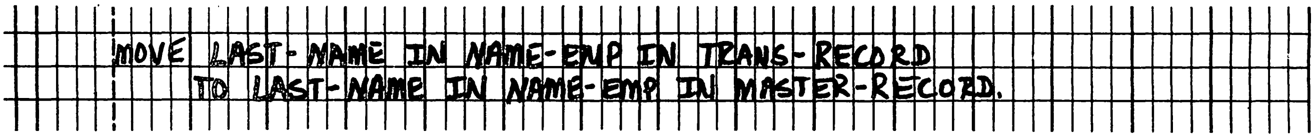
```
MOVE NAME-EMP IN MASTER-RECORD
    TO NAME-EMP IN TRANS-RECORD.
```

Figure 13-16 Example of Move Statement with Name Qualification

Note in the example in Figure 13-16 that the field to be moved is identified as NAME-EMP IN MASTER-RECORD. This is known as name qualification because NAME-EMP is used for more than one field within the program. The term "IN MASTER-RECORD", however, uniquely identifies which NAME-EMP field is being referenced. Thus, the compiler will be able to determine that the NAME-EMP field within the group item MASTER-RECORD is the sending field in the MOVE statement.

Similarly, the term "NAME-EMP IN TRANS-RECORD" identifies the receiving field in the Move statement. It should be noted that the word "OF" is logically the same as the word "IN" when identifying qualified names. Thus, the data-name NAME-EMP OF TRANS-RECORD would also identify the NAME-EMP field in the group item TRANS-RECORD as the receiving field in the Move statement.

It must be noted that whenever the same data-names are used to define different areas in main storage, they must never be used as identifiers within an imperative statement without some qualification which will make them unique. It should also be noted that more than one level of qualification may be used. The following example illustrates this.



```
MOVE LAST-NAME IN NAME-EMP IN TRANS-RECORD
    TO LAST-NAME IN NAME-EMP IN MASTER-RECORD.
```

Figure 13-17 Example of Multi-Level Name Qualification

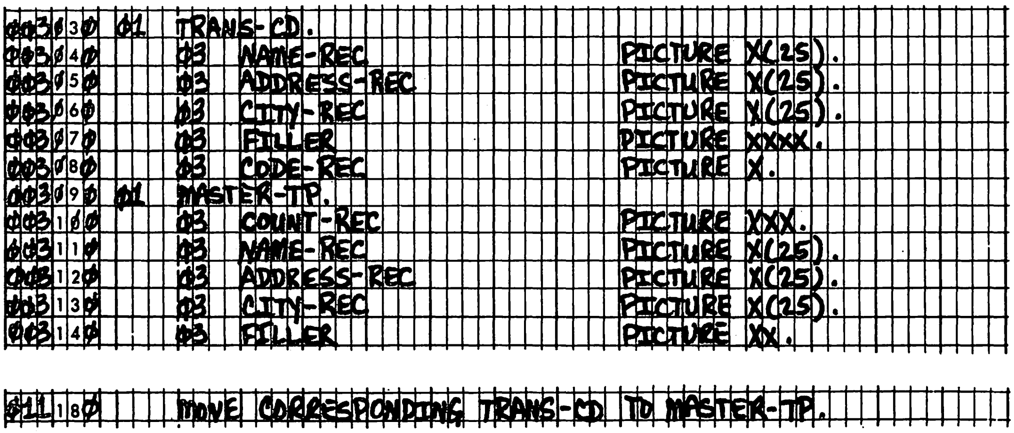
Note in Figure 13-17 that there are two names, NAME-EMP and TRANS-RECORD qualifying the data-name LAST-NAME for the sending field. The only requirement when specifying qualified names is that the name be qualified to a point where it is unique and can be identified by the COBOL compiler.

MOVE CORRESPONDING

In the previous example of name qualification, it was seen that the same data-name could be used for an elementary data item as long as it belonged to a group item with a unique name. It should be noted also that each MOVE statement which used name-qualification referenced only one data item within the group.

The CORRESPONDING option may be used to reference all fields with common data names within two different groups. The following example illustrates the use of the CORRESPONDING option when used with the MOVE statement.

EXAMPLE



```
003030 01  TRANS-CD.
003040         03  NAME-REC                   PICTURE X(25).
003050         03  ADDRESS-REC                PICTURE X(25).
003060         03  CITY-REC                   PICTURE X(25).
003070         03  FILLER                     PICTURE XXXX.
003080         03  CODE-REC                   PICTURE X.
003090 01  MASTER-TP.
003100         03  COUNT-REC                  PICTURE XXX.
003110         03  NAME-REC                   PICTURE X(25).
003120         03  ADDRESS-REC                PICTURE X(25).
003130         03  CITY-REC                   PICTURE X(25).
003140         03  FILLER                     PICTURE XX.

011180     MOVE CORRESPONDING TRANS-CD TO MASTER-TP.
```

BEFORE MOVE

TRANS-CD		MASTER-TP	
NAME-REC	ALFRED R. JOHONSKI	COUNT-REC	275
ADDRESS-REC	7245 DRUMMOND AVE	NAME-REC	JAMES JILLIAN
CITY-REC	NORTH LAKE, NEW YORK	ADDRESS-REC	1927 HIWAY 37
CODE-REC	3	CITY-REC	BRUSHVILLE, TEXAS

AFTER MOVE

TRANS-CD		MASTER-TP	
NAME-REC	ALFRED R. JOHONSKI	COUNT-REC	275
ADDRESS-REC	7245 DRUMMOND AVE	NAME-REC	ALFRED R. JOHONSKI
CITY-REC	NORTH LAKE, NEW YORK	ADDRESS-REC	7245 DRUMMOND AVE
CODE-REC	3	CITY-REC	NORTH LAKE, NEW YORK

Figure 13-18 Example of MOVE CORRESPONDING

The general format of the MOVE CORRESPONDING statement is shown below.

```
       ( CORRESPCNDING )
MOVE  <                 >  identifier-1 TO identifier-2
       ( CORR          )
```

Figure 13-19 General Format of MOVE CORRESPONDING Statement

As can be seen, the verb MOVE must be stated the same as in MOVE statements illustrated previously. The word CORRESPONDING or the abbreviation CORR must immediately follow the MOVE verb separated by one or more blanks.

Identifier-1 is used to specify a group-item data name which contains the sending fields to be moved in the MOVE statement. Identifier-2 specifies the group-item data name which contains the receiving fields. When the MOVE CORRESPONDING instruction is executed, as illustrated in Figure 13-18, all data stored in fields within the group-item specified by Identifier-1 with the same data names as data stored in the fields within the group-item specified by Identifier-2 are moved. Thus, for example, since the name NAME-REC appears both in the group specified by Identifier-1 and in the group specified by Identifier-2, the data in the field NAME-REC is moved from the sending group-item (TRANS-CD) to the receiving group-item (MASTER-TP). The same is true for the fields ADDRESS-REC and CITY-REC, which are specified in both the TRANS-CD group and the MASTER-TP group. It should be noted that the individual names need not be specified in order to move the data from one field to another. The use of the CORRESPONDING Entry causes these moves to take place by merely specifying the group-item data names.

It should be noted also that names within either group which do not have a corresponding name in the other group are not affected by the Move Corresponding statement. Thus, in the example in Figure 13-18, the field CODE-REC, for which there is not a corresponding name in the MASTER-TP group, is not moved and the field COUNT-REC in the MASTER-TP group is not altered because there is no correspondingly-named field in the TRANS-CD group. Any FILLERs which are defined in either of the group-items are not moved even though the names are the same.

The CORRESPONDING option may be used with other instructions as well as the MOVE instruction. For example, it may be used with the ADD statement in order to add the values in fields with corresponding names within a group item. It must be remembered, however, that when the ADD CORRESPONDING statement is utilized, all of the fields which are to be added within the group item must be validly signed numeric fields the same as when the simple ADD statement is used or there may be a cancellation of the program or other unpredictable results.

PROGRAMMING ASSIGNMENT

CASE STUDY

The final programming assignment incorporates many of the concepts studied in the previous chapters. Upon the solution of this problem the student should be able to solve a wide variety of business programming problems.

INSTRUCTIONS

The Weekly Service Contract report illustrated below is currently being prepared manually. This report indicates the number and amount of Service Contracts sold and the percent of the Service Contracts charged to a PARTS Account and the percent charged to LABOR Account. For example, in Store 851, Department 9, three service contracts were sold (UNITS) in the amount of $22.00. Thirty percent of the total amount is charged to the Parts Account ($6.60) and the balance is charged to the Labor Account ($15.40). There were no renewals in Department 9. The totals on the right hand side of the report are the sum of the Contracts Sold and Renewals.

Intermediate totals are taken when there is a change in Store Number, and final totals are taken after all cards have been processed.

DATE 04/20/70 WEEKLY SERVICE CONTRACT REPORT PAGE 01

STORE	DEPT.	PER CENT	CONTRACTS SOLD				RENEWALS				TOTALS			
			PARTS	LABOR	UNIT	AMOUNT	PARTS	LABOR	UNIT	AMOUNT	PARTS	LABOR	UNIT	AMOUNT
851	9	30%	6.60	15.40	3	22.00					6.60	15.40	3	22.00
	22	20%	6.98	27.92	2	34.90					6.98	27.92	2	34.90
	26	20%	125.82	503.28	18	629.10	14.95	59.80	5	74.75	140.77	563.08	23	703.85
	32	30%	3.00	7.00	1	10.00					3.00	7.00	1	10.00
	42	15%	1.50	8.50	1	10.00					1.50	8.50	1	10.00
	46	17%	7.62	37.23	3	44.85	4.92	24.03	2	28.95	12.54	61.26	5	73.80
	47	12%	1.43	10.52	1	11.95	1.55	11.40	1	12.95	2.98	21.92	2	24.90
	57	5%	21.77	413.53	14	435.30	9.80	186.15	10	195.95	31.57	599.68	24	631.25
	65	30%	5.99	13.96	1	19.95	4.49	10.46	1	14.95	10.48	24.42	2	34.90
	TOTALS	.....	$180.71*	$1,037.34*	44	$1,218.05*	$35.71*	$291.84*	19	$327.55*	$216.42*	$1,329.18*	63	$1,545.60*
852	13	17%	21.76	106.24	2	128.00					21.76	106.24	2	128.00
	24	15%	13.97	79.13	1	93.10					13.97	79.13	1	93.10
	46	17%	6.81	33.25	3	40.06					6.81	33.25	3	40.06
	TOTALS	.....	$42.55*	$218.62*	6	$261.16*	$.00*	$.00*	0	$.00*	$42.54*	$218.62 *	6	$261.16*
FINAL	TOTALS	....	$223.25**	$1,255.96**	50	$1,479.21**	$35.71**	$291.84**	19	$327.55**	$258.96**	$1,547.80**	69	$1,806.76**

CARD FORMATS

It has been decided to convert this application to a computer prepared report. The card format for each transaction is given below.

Card Columns 15-17: Store Number
Card Columns 57-58: Department Number
Card Columns 70-76: Amount
Card Column 80: Control Code
1 = Contract Sold
2 = Renewal

There is one card for each Service Contract sold and one card for each renewal.

It will be noted that the percentage to be applied to the parts and labor accounts is not contained on the input cards. These percentages are to come from cards which are to be used to build a table for each department. The format of the cards to build the table are:

Columns 2-3: Department Number
Columns 5-7: Percentage for Parts Account (XX.X%)

These cards should be read prior to reading the detail data and a table should be established for each department number.

OUTPUT

The printer spacing chart should be prepared by the programmer to include all of the information shown on the hand-prepared report on the previous page.

Test Data

The data which can be used to load the department-percentage table is illustrated below.

DEPARTMENT NUMBER	PARTS PERCENTAGE
09	30.0
13	17.0
18	23.0
22	20.0
24	15.0
26	20.0
32	30.0
42	15.0
46	17.0
47	12.0
49	34.6
57	05.0
65	30.0
69	18.0

Suggested test data to be used for the testing of the program is illustrated below.

```
          STORE              DEPT                                       AMOUNT           CODE

           851                09                                        0000350            1
           851                09                                        0000450            1
           851                09                                        0001400            1
           851                22                                        0001490            1
           851                22                                        0002000            1
           851                26                                        0000475            1
           851                26                                        0002945            1
           851                26                                        0003150            1
           851                26                                        0003200            1
           851                26                                        0003330            1
           851                26                                        0003390            1
           851                26                                        0003600            1
           851                26                                        0003794            1
           851                26                                        0003800            1
           851                26                                        0004080            1
           851                26                                        0004500            1
           851                26                                        0005892            1
           851                26                                        0006055            1
           851                26                                        0002295            1
           851                26                                        0002720            1
           851                26                                        0003000            1
           851                26                                        0003189            1
           851                26                                        0003495            1
           851                26                                        0001222            2
           851                26                                        0001432            2
           851                26                                        0001891            2
           851                26                                        0002107            2
           851                26                                        0000823            2
           851                32                                        0001000            1
           851                42                                        0001000            1
           851                46                                        0000706            1
           851                46                                        0001483            1
           851                46                                        0002296            1
           851                46                                        0001395            2
           851                46                                        0001500            2
           851                47                                        0001195            1
           851                47                                        0001295            2
           851                57                                        0000999            1
           851                57                                        0001000            1
           851                57                                        0001332            1
           851                57                                        0001625            1
           851                57                                        0001986            1
           851                57                                        0002342            1
           851                57                                        0002451            1
           851                57                                        0003150            1
           851                57                                        0003222            1
           851                57                                        0003841            1
           851                57                                        0004083            1
           851                57                                        0004533            1
           851                57                                        0005968            1
           851                57                                        0006998            1
           851                57                                        0000895            2
           851                57                                        0001125            2
           851                57                                        0001382            2
           851                57                                        0001445            2
           851                57                                        0001529            2
           851                57                                        0001896            2
           851                57                                        0002103            2
           851                57                                        0002448            2
           851                57                                        0003030            2
           851                57                                        0003742            2
           851                65                                        0001995            1
           851                65                                        0001495            2
           852                13                                        0005207            1
           852                13                                        0007593            1
           852                24                                        0009310            1
           852                46                                        0001003            1
           852                46                                        0001475            1
           852                46                                        0001528            1

       TOTAL CARDS   69
```

APPENDIX A

INTERNAL DATA REPRESENTATION

INTERNAL DATA REPRESENTATION

Most computer systems currently in use utilize MAGNETIC CORE STORAGE. A magnetic core is a very small ring of ferromagnetic material, a few hundredths of an inch in outside diameter. Cores are placed on a wire and an electrical current is sent through the wire. If enough current passes through, the cores become magnetized. Reversing the direction of the current through the wire changes the magnetic state or polarity of the core. Thus, the direction of the magnetic charge, or polarity of the core, can be used to represent an "on" or "off" condition or a "0" or "1". It is this basic concept that provides the basis for the storing of information in the Central Processing Unit of a computer system.

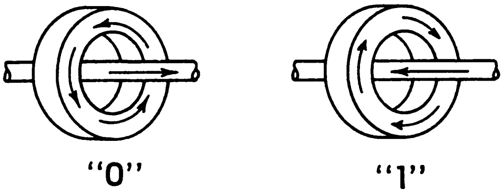


Figure A-1 Magnetic Cores

Because computer systems store information by means of magnetic core storage in which there are only two possible states, "0" or "1", the system is said to operate in a binary mode. To understand the logical structure of recording data utilizing this method requires a basic understanding of the binary number system.

DECIMAL NUMBER SYSTEM

A review of the "place value" concept as used with the decimal number system is of value in understanding the structure of the binary number system.

In the decimal number system each place position from right to left is as follows: units, tens, hundreds, thousands, etc. This is obtained by using a base of 10 (as there are ten symbols 0-9 in this decimal system) and raising this base to the next highest power each position to the left.

The following chart illustrates the place values of the first four positions of the decimal number system.

THOUSANDS	HUNDREDS	TENS	UNITS	Place Value
10^3	10^2	10^1	10^0	Base

Figure A-2 Decimal Number System Place Value Chart

Thus, the decimal number 1110 when analyzed means:

1 – One thousand
1 – One hundred
1 – Ten
0 – Units

or one thousand one hundred and ten.

1	1	1	0	
THOUSANDS	HUNDREDS	TENS	UNITS	Place Value
10^3	10^2	10^1	10^0	Base

Figure A-3 Representation of the Decimal Number 1110

BINARY NUMBER SYSTEM

In the binary number system 0 and 1 are the only digits used. With the use of these two digits any number may be represented by use of the positional notation concept.

The binary number system uses a base of 2 with each place position assigned the following values: one, two, four, eight, sixteen, etc. (values double each place position to the left). The base of 2 is used as there are only two symbols in the binary number system (0-1). The place value is established by raising the base of 2 to the next highest power each place position to the left.

The following chart illustrates the place values of the first four positions in a binary number system.

EIGHT	FOUR	TWO	ONE	Place Value
2^3	2^2	2^1	2^0	Base

Figure A-4 Binary Number System Place Value Chart

Thus, the binary number 10 is equal to the decimal number 2.

1 - two
0 - units or ones

0	0	1	0	
EIGHT	FOUR	TWO	ONE	Place Value
2^3	2^2	2^1	2^0	Base

Figure A-5 Representation of the Binary Number 10

The number three in binary would be:

0	0	1	1	
EIGHT	FOUR	TWO	ONE	Place Value
2^3	2^2	2^1	2^0	Base

Figure A-6 Representation of the Binary Number 0011

Thus, 11 in binary is equal to the number three.

The following is a summary of the binary equivalents of the decimal numbers 0-9.

Decimal	*Binary*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Figure A-7 Decimal and Binary Numbers 0-9

In most computer systems magnetic core storage is designed to utilize a binary coding structure. In this type of storage organization cores are assigned the values of 1, 2, 4, and 8. Cores are placed on a matrix of wires. At the point where two impulsed wires intersect, the current is great enough to change the polarity of a core and thus alter the information stored.

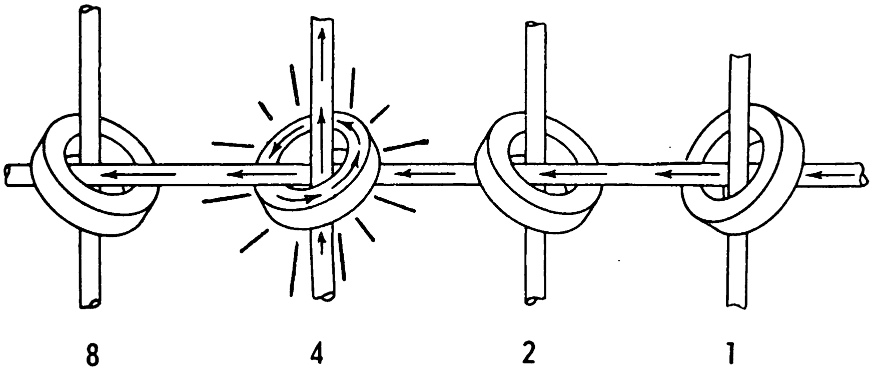


Figure A-8 Magnetic Core Storage

Thus, using magnetic cores as a form of storage with each core capable of representing either a "0" or a "1", and combining this concept with the place values of the binary number system it is possible to develop a mathematically logical method of representing numerical values in magnetic core storage. For example, by the use of four magnetic cores operating as one unit of information, it is possible to represent the digits 0 through 9 by turning "ON" the proper combination of cores.

The following diagram illustrates how the digits 0-9 could be represented in core storage using a binary type of coding structure.

0= ○○○○
1= ○○○●
2= ○○●○
3= ○○●●
4= ○●○○
8 4 2 1

5= ○●○●
6= ○●●○
7= ○●●●
8= ●○○○
9= ●○○●
8 4 2 1

● = Core "On"
○ = Core "Off"

Figure A-9 Representation of the Decimal Values 0-9 in Core Storage

Information entered into the storage unit of the central processing unit of a computer system is recorded in specific core storage locations which are "addressable". The data placed in these addressable locations can thus be processed or moved internally in storage as required. It should be noted that the series of instructions used to process the data are also stored in the storage unit through the use of magnetic cores.

EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE

A variety of coding methods have been used in computer systems to represent numbers, letters of the alphabet and special characters. The coding structure used on many systems is known as the Extended Binary Coded Decimal Interchange Code or EBCDIC. This coding technique provides a means of storing letters of the alphabet, numbers, or special characters internally in a series of 8 magnetic cores or "bits" which totally are referred to as a byte.

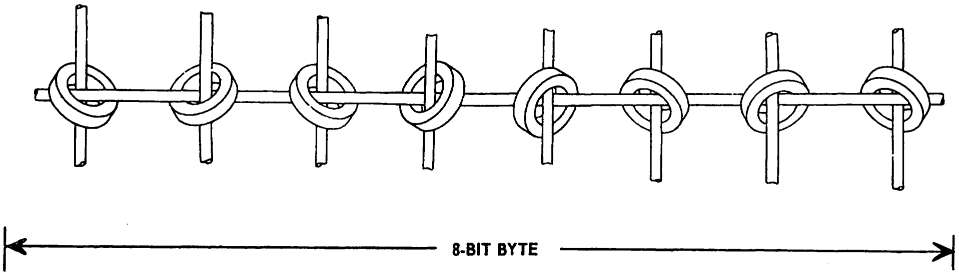


Figure A-10 Core Representation of the Eight-Bit Byte

Each byte of storage has a unique address associated with it so that the programmer can selectively locate or retrieve a unit of information placed in storage. The "storage locations" are numbered from zero up to the size of storage available for a particular system. Most computers provide for a storage capacity of from 8,192 bytes of storage to 6,291,456 bytes of storage. The storage capacity of models are normally referred to as 8K, 32K, 64K, etc., meaning 8 thousand, 32 thousand, 64 thousand, etc., bytes of storage. "Typical" systems range from 8K to 512K bytes of storage.

Zoned Decimal Format

When one character is represented in storage by one addressable byte it is said to be stored in the Zoned Decimal Format. For purposes of referencing the individual cores or bits in a byte, each bit position is numbered as follows:

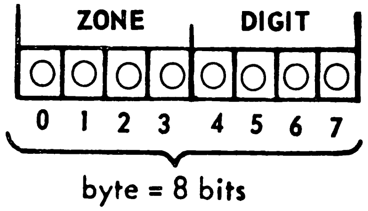
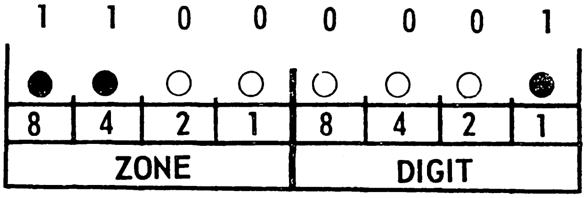


Figure A-11 Schematic Representation of Eight-Bit Byte

Thus, it is possible to refer to bits 0-3 when referencing the four leftmost bit positions of the byte, or bits 4-7 when referencing the four rightmost bit positions of the byte.

When a character is stored in the byte, a particular binary value is placed in each of the four bits of the byte and together these eight bits of the byte determine the character stored. Bit positions 0-3 of the byte are used to express the ZONE portion of a character and bit positions 4-7 are used to express the DIGIT portion. It should be noted that bit positions 0-3 and 4-7 have a place value of 8, 4, 2, 1 respectively. (Note the relationship to the binary number system.)

The letters of the alphabet A-I have a bit configuration of 1100 (decimal 12) in the zone portion of the byte. The digit position would contain the decimal values of 1-9 for A-I respectively. Thus, the letter of the alphabet A would be represented in the byte as:



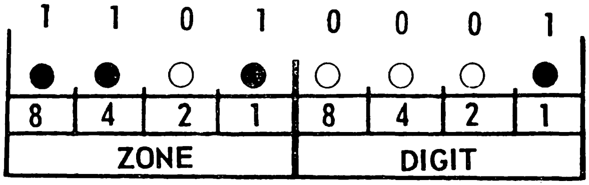
= Decimal Value 12–1

Note: 1 = Bit is "ON"
0 = Bit is "OFF"

Figure A-12 Representation of the Letter of the Alphabet A

The letters of the alphabet J-R have a bit configuration of 1101 (decimal 13) in the zone portion of the byte. The digit portion of the byte would contain the decimal values 1-9 for J-R respectively. Thus, the letter of the alphabet J would appear as follows:

J =



= Decimal Value 13–1

Figure A-13 Representative of the Letter of the Alphabet J

The letters of the alphabet S-Z have a bit configuration of 1110 (decimal 14) in the zone portion of the byte. The digit portion of the byte would contain the decimal values 2-9 for S-Z respectively. Thus, the letter of the alphabet S would appear as follows:

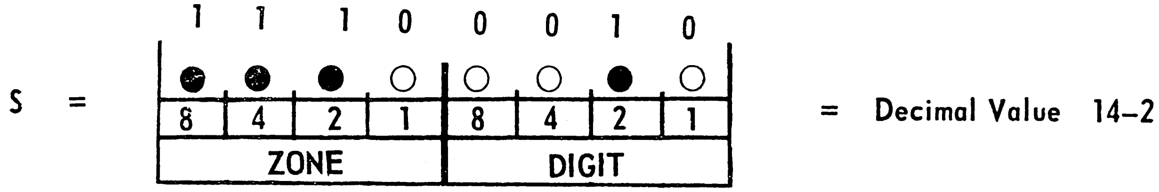

1 1 1 0 0 0 1 0
S =
8 4 2 1 8 4 2 1
ZONE
DIGIT
= Decimal Value 14-2

Figure A-14 Representation of the Letter of the Alphabet S

The following diagram illustrates the bit configurations for the letters of the alphabet A-Z.

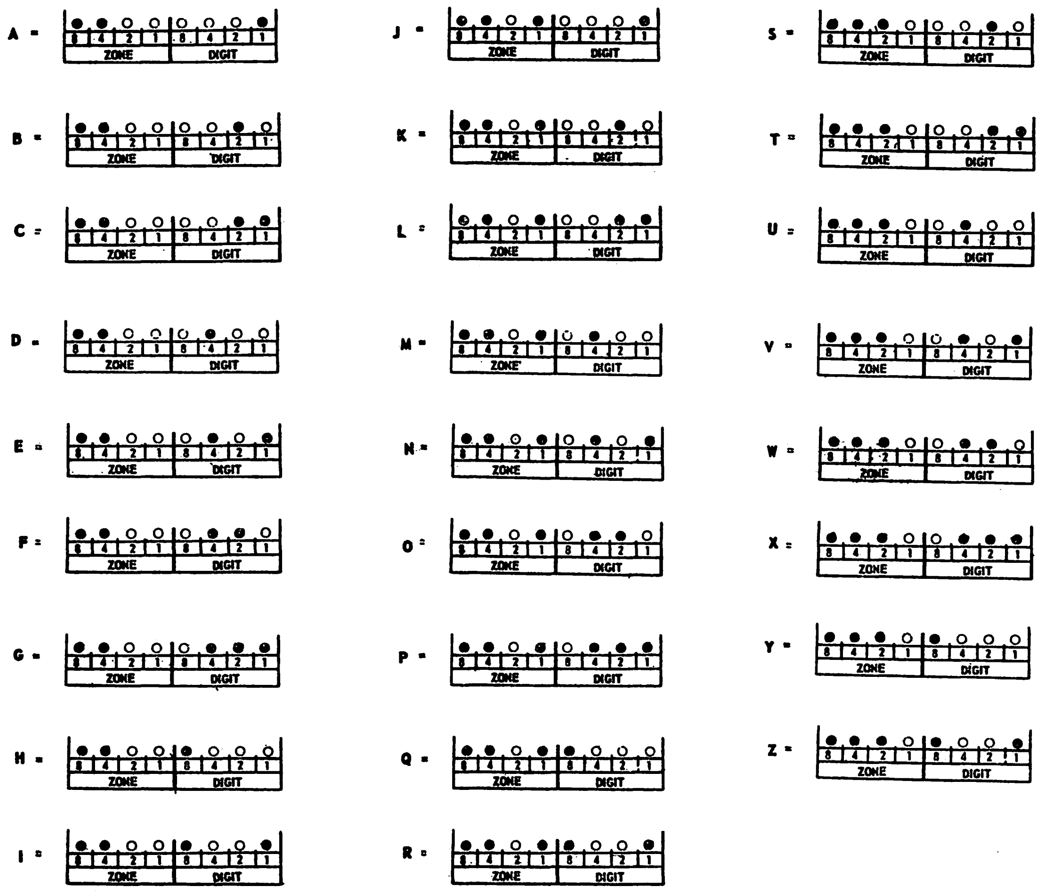

A =
B =
C =
D =
E =
F =
G =
H =
I =
J =
K =
L =
M =
N =
O =
P =
Q =
R =
S =
T =
U =
V =
W =
X =
Y =
Z =
8 4 2 1 8 4 2 1
ZONE
DIGIT

Figure A-15 Letters of the Alphabet A-Z in the Byte

Special characters also may be represented using the zoned decimal format. The following illustrates the representation of the "$" and "?".

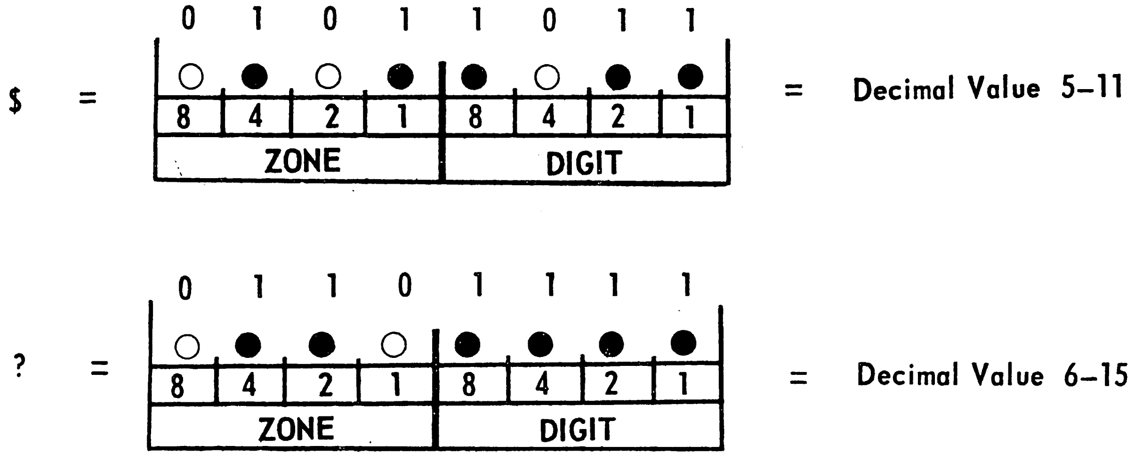


Figure A-16 Representation of Special Characters

The following chart illustrates the bit configuration for the letters of the alphabet and frequently used special characters.

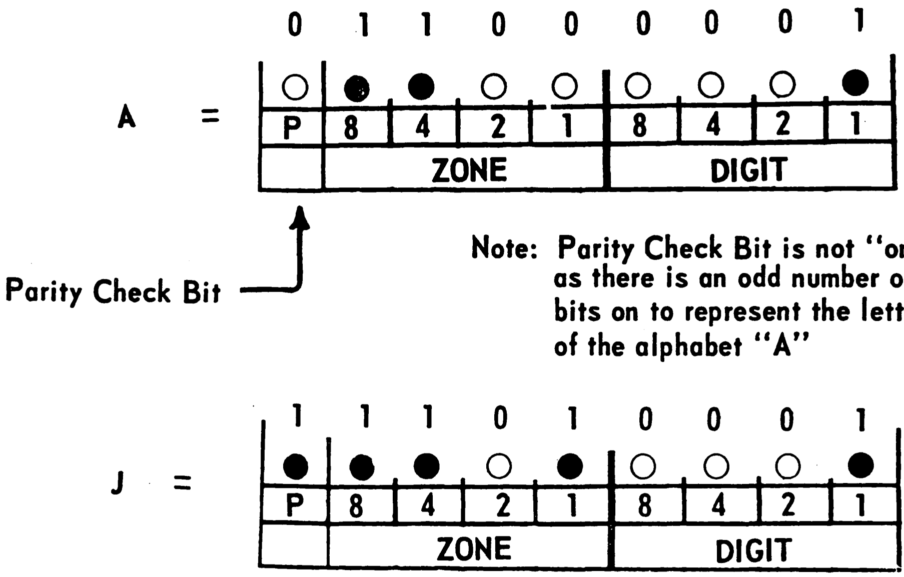
EBCDIC Code	EBCDIC Graphic	EBCDIC Code	EBCDIC Graphic
1100 0001	A	1101 1001	R
1100 0010	B	1110 0010	S
1100 0011	C	1110 0011	T
1100 0100	D	1110 0100	U
1100 0101	E	1110 0101	V
1100 0110	F	1110 0110	W
1100 0111	G	1110 0111	X
1100 1000	H	1110 1000	Y
1100 1001	I	1110 1001	Z
1101 0001	J	0101 1011	$ (dollar sign)
1101 0010	K	0101 1100	* (asterisk)
1101 0011	L	0110 1111	? (question mark)
1101 0100	M	0111 1010	: (colon)
1101 0101	N	0111 1011	# (number sign)
1101 0110	O	0111 1100	@ (at the rate of)
1101 0111	P	0111 1101	' (apostrophe)
1101 1000	Q		

Figure A-17 Summary of Representation of Letters and Special Characters

Parity Check

It should be noted that there is also an internal hardware check bit associated with each byte. When data is read into the system, a PARITY CHECK BIT will automatically go on if required. Some computers require that an odd number of bits always be "on" and others require an even number of bits to be "on".

In the following examples, note the use of the parity bit to maintain "odd parity".



Note: Parity Check Bit is not "on" as there is an odd number of bits on to represent the letter of the alphabet "A"

Note: Parity Check Bit is "on" to maintain an odd number of bits "on" in the byte.

Figure A-18 Use of the Parity Check Bit

When data is moved internally or other operations occur, the machine automatically checks each position for odd parity. If odd parity does not exist (the machine "drops a bit"), an error condition is indicated to the operator. (Future diagrams of the byte will not illustrate the use of the parity bit but it should be noted that each byte of storage contains a check bit.)

NUMERIC REPRESENTATION

The general form of data stored in the zoned decimal format is as follows:

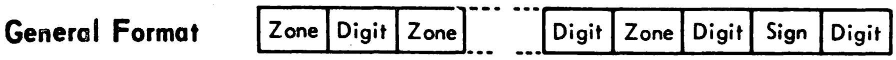


Figure A-19 General Form-Zoned Decimal Format

Note that bits 0-3 of the rightmost byte of a field are used for sign control.

When positive numbers are read into core, the zone portion of the byte (including the rightmost byte) is filled with 1's. This technique makes numbers higher than letters of the alphabet in a collating sequence.

The following examples illustrate the representation of the numbers 1-9 in the zoned decimal format.

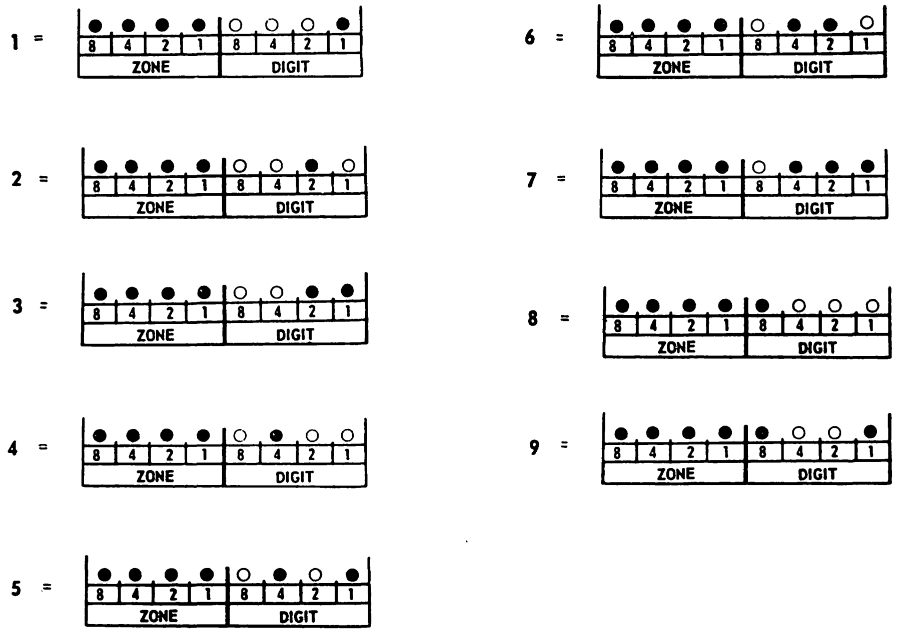


Figure A-20 Representation of the Numbers 1-9 in the Byte

In a binary type notation the numbers 1-9 could be illustrated as follows:

Decimal	EBCDIC[*] CODE
1	1111 0001
2	1111 0010
3	1111 0011
4	1111 0100
5	1111 0101
6	1111 0110
7	1111 0111
8	1111 1000
9	1111 1001

[*] Extended Binary-Coded-Decimal Interchange Code.

Sign Control

A "Quantity" field in card columns 79-80 read from a card could be stored in bytes 10000 and 10001 as follows: (It should be noted bytes 10000 and 10001 were selected arbitrarily to illustrate the concept of an addressable byte.)

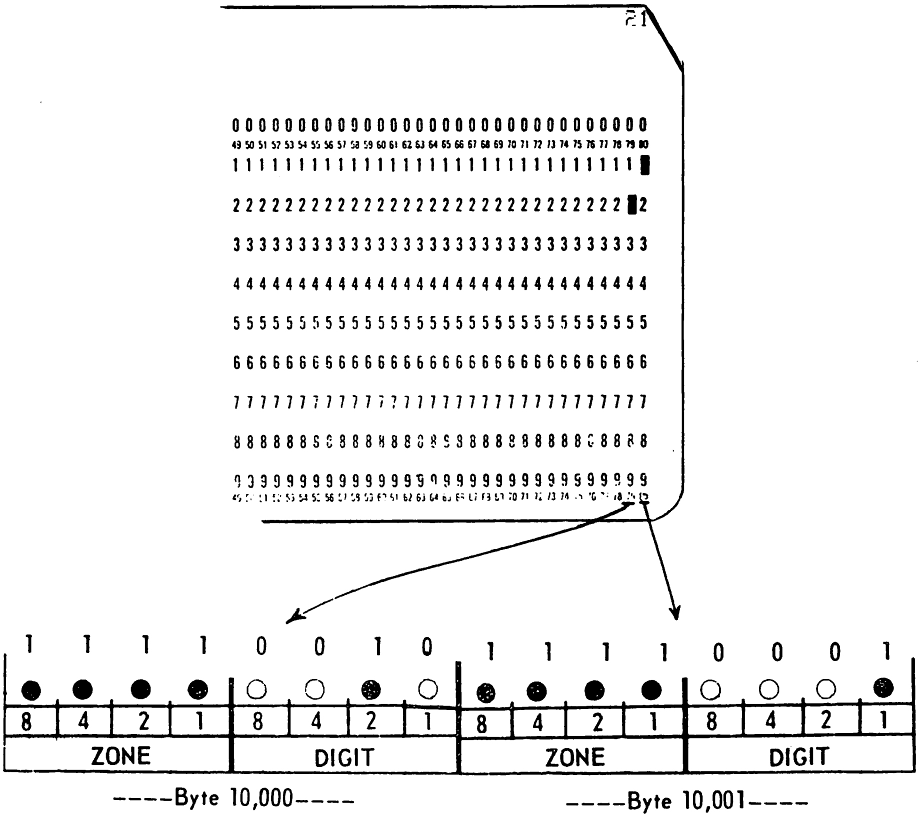
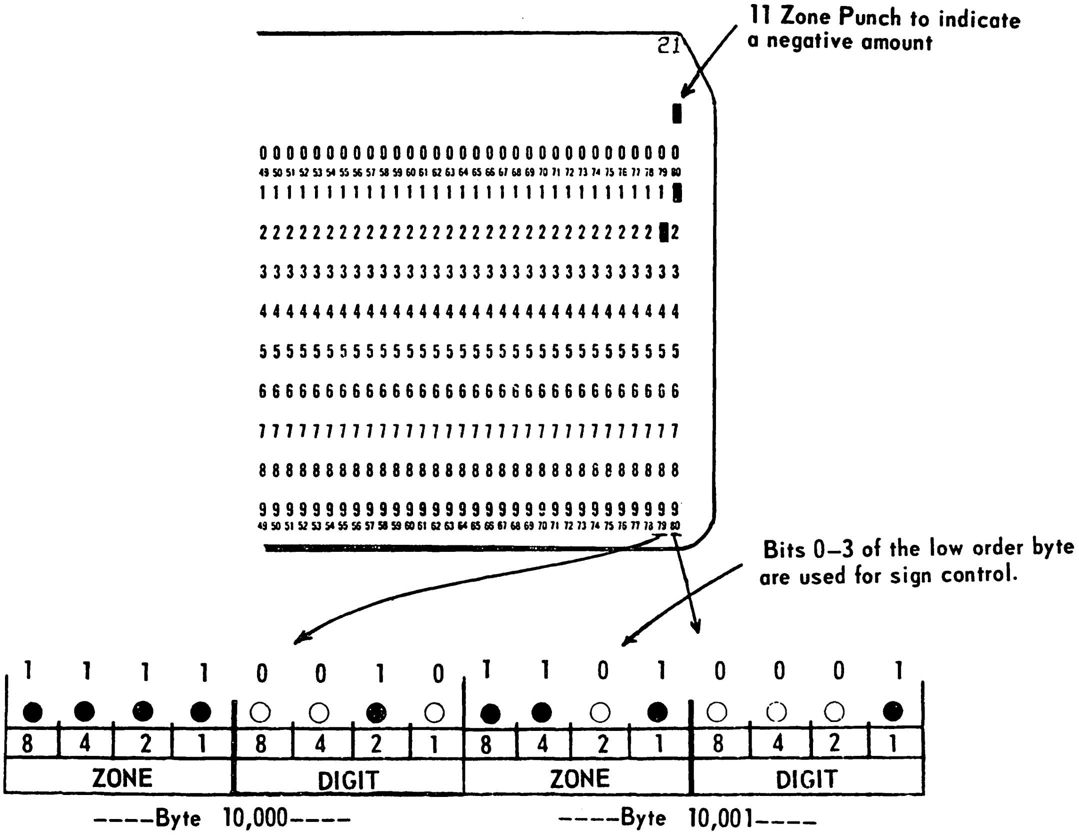

1 1 1 1 0 0 1 0 1 1 1 1 0 0 0 1
8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1
ZONE
DIGIT
ZONE
DIGIT
----Byte 10,000----
----Byte 10,001----

Figure A-21 Representation of a Positive Quantity Field

Note that bits 0-3 of the rightmost byte contains 1111. The bit configuration of 1111 is used as a + sign. Thus, the Quantity Field in bytes 10000 and 10001 is represented as a positive value.

On a punched card, an 11 zone punch in the low-order position of the field is used to indicate a minus or negative amount. The following example illustrates how a negative amount would appear in storage in bytes 10000 and 10001.



11 Zone Punch to indicate a negative amount
21
Bits 0–3 of the low order byte are used for sign control.
1 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1
8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1
ZONE
DIGIT
ZONE
DIGIT
----Byte 10,000----
----Byte 10,001----

Figure A-22 Representation of a Negative Quantity Field

Note that the sign in bits 0-3 of the low-order byte contains the bit configuration 1101. A 1101 in bits 0-3 represents a minus sign. Thus, the Quantity Field 21 stored in bytes 10000 and 10001 is considered to be a negative amount.

PACKED DECIMAL FORMAT

On some machines, for example the System/360 and System/370, data must be converted from the Zoned Decimal Format to a PACKED DECIMAL format when performing arithmetic operations. The general form of the packed decimal format is below:

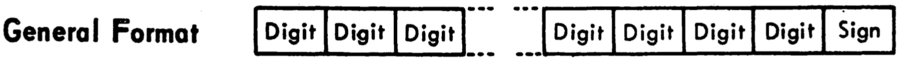

General Format
Digit
Digit
Digit
Digit
Digit
Digit
Digit
Sign

Figure A-23 General Form – Packed Decimal Format

As can be seen from the above illustration, two decimal digits are stored in a single byte with the sign in bits 4-7 of the low-order byte.

The following diagram illustrates a card read into core and stored in the zoned decimal format and the same field after it has been "packed".

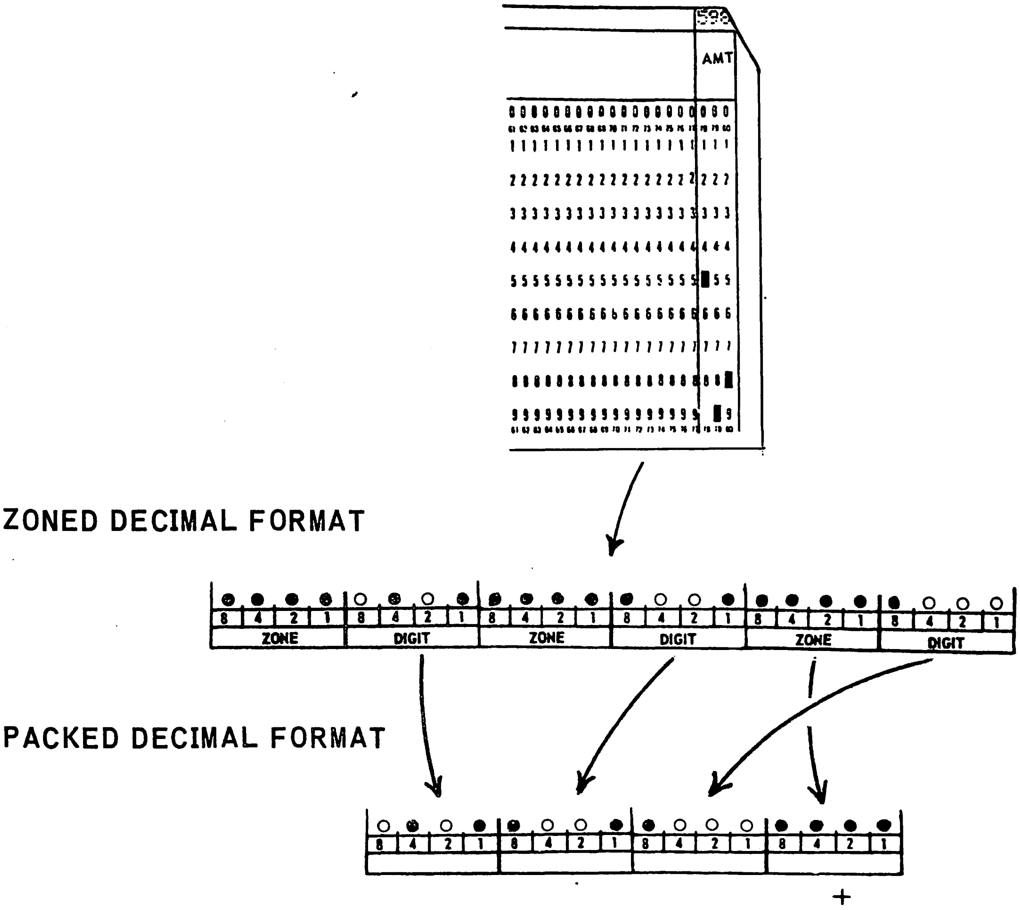

AMT
ZONED DECIMAL FORMAT
8 4 2 1
ZONE
DIGIT
PACKED DECIMAL FORMAT
+

Figure A-24 Steps in Packing a Field

It should be noted that this packed-decimal format is specified by the USAGE IS COMPUTATIONAL-3 entry for the data item.

After arithmetic operations have been performed on data, a positive number will contain a sign bit configuration of 1100. The following example illustrates the number +598 in the packed decimal format with a sign bit configuration of 1100.

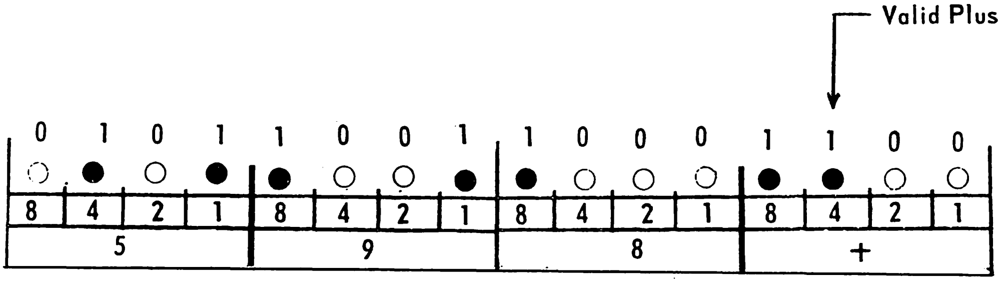


Figure A-25 Representation of Positive Amount After a Calculation

Thus, valid + signs include the bit configuration of 1111 and also 1100. Before data can be printed out the bit configuration 1100 must be changed back to 1111 however.

When data is recorded in packed decimal format, one-half of a byte cannot be used; therefore, when a number such as + 22 is recorded, bits 0-3 of the leftmost byte contain zeroes.

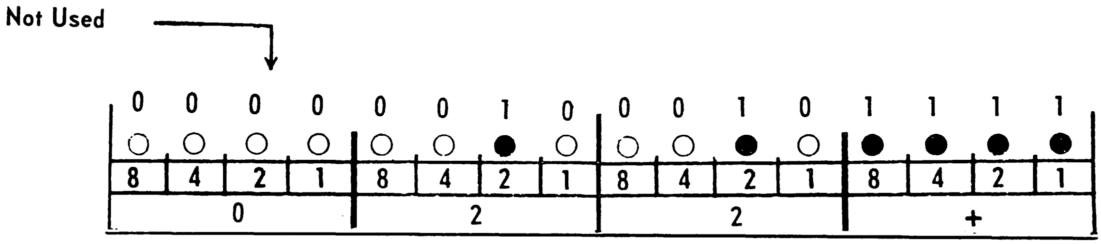


Figure A-26 Representation of the storage of a Field with an Even Number of Digits

It must again be emphasized that when data is read into a system from a punched card, it is recorded in the Zoned Decimal Format; however, before arithmetic operation can be performed the data must be converted to a packed form when processing on a System/360 or System/370.

HEXADECIMAL NUMBER SYSTEM

As explained previously, data is recorded by means of a series of magnetic cores being "Off" or "On" which may be represented by "0's" or "1's". Data recorded in this form is difficult to work with, however. A "core dump" in this binary form in which the contents of each of the four bits in the byte is printed out might appear as follows:

111110011111010111110011 . . . . . etc.

To facilitate the reading of data in a binary form using EBCDIC, hexadecimal notation is used.

The hexadecimal system represents the decimal values 0-15 by means of 16 individual symbols. The representation of the digits 0-9 are the same in the hexadecimal number system as in the decimal number system; however, to represent the decimal quantities 10-15, the hexadecimal system uses the letters of the alphabet A-F. Thus, the decimal quantity 10 is represented in a hexadecimal as "A". The decimal quantity 11, as hexadecimal "B" etc.

DECIMAL VALUE	HEXADECIMAL NOTATION
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Figure A-27 Decimal and Hexadecimal Representation of Numbers 0-15

As previously illustrated, a core dump in binary form would appear as a series of 0's and 1's.

```
11110101111110110111101111111011
```

This type of notation is difficult for programmers to read; therefore, the systems utilizing EBCDIC normally use the hexadecimal notation. With hexadecimal notation every four bits represent a hexadecimal number. Thus, if the binary notation (0's and 1's) is separated into 4 bits and converted to hexadecimal notation, it would appear as illustrated.

Binary	1111	0101	1111	0110	1111	0111	1111	0011
Hex	F	5	F	6	F	7	F	3
Decimal	5		6		7		3	

Figure A-28 Binary, Hexadecimal, and Decimal Representation of Numeric Values

It can be seen that the original binary representation converted to hexadecimal notation is F5F6F7F3. Thus, the F's (1111) imply numeric data is recorded in the zoned decimal format. The data in decimal form is thus +5673.

An example of alphabetic data in storage is as follows:

110000011100001011000011

Again it can be seen that pure binary notation is difficult to read. By converting this string of 0's and 1's to a hexadecimal notation the form becomes:

Binary	1100	0001	1100	0010	1100	0011
Hex	C	1	C	2	C	3
Alpha	A		B		C	

Figure A-29 Binary, Hexadecimal, and Alphabetic Representation of Data

By noting that the hexadecimal C is used as the zone portion of the letters of the alphabet A-I it can be determined that C1C2C3 in hexadecimal is actually the alphabetic data ABC represented internally in the byte.

The following chart summarizes the representation of the numbers 0 - 9 and the letters of the alphabet A - Z in a binary and a hexadecimal form.

Graphics	Binary	Hexadecimal
0	1111 0000	F0
1	1111 0001	F1
2	1111 0010	F2
3	1111 0011	F3
4	1111 0100	F4
5	1111 0101	F5
6	1111 0110	F6
7	1111 0111	F7
8	1111 1000	F8
9	1111 1001	F9
A	1100 0001	C1
B	1100 0010	C2
C	1100 0011	C3
D	1100 0100	C4
E	1100 0101	C5
F	1100 0110	C6
G	1100 0111	C7
H	1100 1000	C8
I	1100 1001	C9
J	1101 0001	D1
K	1101 0010	D2
L	1101 0011	D3
M	1101 0100	D4
N	1101 0101	D5
O	1101 0110	D6
P	1101 0111	D7
Q	1101 1000	D8
R	1101 1001	D9
S	1110 0010	E2
T	1110 0011	E3
U	1110 0100	E4
V	1110 0101	E5
W	1110 0110	E6
X	1110 0111	E7
Y	1110 1000	E8
Z	1110 1001	E9

Figure A - 30 Summary of Data Representation in Hexadecimal Form

Hexadecimal Number System — Place Value

A hexadecimal number system is comprised of 16 symbols representing the decimal digits 0-15. The hexadecimal system uses the decimal value of 16 as its base. By raising a base of 16 to the next highest power moving to the left, the place values of 1, 16, 256, 4096, etc. are established. The following diagram illustrates a place value chart for the first four positions of a hexadecimal number system.

4096	256	16	1	Place Value
16^3	16^2	16^1	16^0	Base

Figure A-31 Place Value Chart For the Hexadecimal Number System

The decimal value 16 is represented in hexadecimal as 10; that is, there is "1" sixteen, and "0" ones. See Figure A-32 below.

		1	0	
4096	256	16	1	Place Value
16^3	16^2	16^1	16^0	Base

Figure A-32 Representation of Hexadecimal 16

Thus, 10 in hexadecimal represents the decimal value 16.

The hexadecimal number 110 represents the decimal value 272. Using the place value chart it can be seen that there is "1" two hundred and fifty six, "1" sixteen, and "0" ones, for a total of 272.

	1	1	0	
4096	256	16	1	Place Value
16^3	16^2	16^1	16^0	Base

Figure A-33 Representation of Hexadecimal 110

The chart below illustrates the representation of the decimal numbers 0 -54 in hexadecimal form.

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
18	12
19	13
20	14
21	15
22	16
23	17
24	18
25	19
26	1A
27	1B
28	1C
29	1D
30	1E
31	1F
32	20
33	21
34	22
35	23
36	24
37	25
38	26
39	27
40	28
41	29
42	2A
43	2B
44	2C
45	2D
46	2E
47	2F
48	30
49	31
50	32
51	33
52	34
53	35
54	36

Figure A-34 Summary of Decimal and Hexadecimal Numbers

All machine language instructions, storage locations, and data stored in core are printed out for use by the programmer in a hexadecimal form; therefore, an understanding of the structure of the hexadecimal number system is important for the programmer.

The following is an illustration of "print-out" of storage. Note how data is printed in hexadecimal form.

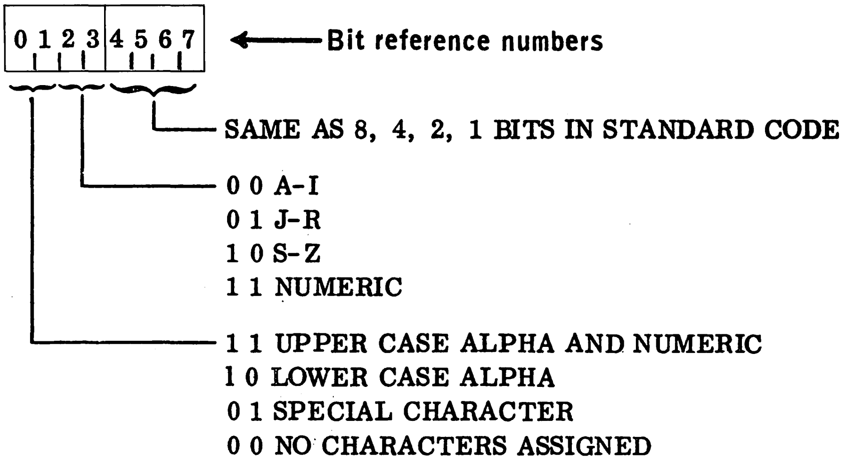
```
0027E0  D5D640D5 C1D4C540 FF150007 FA00292E   0A0407F1 6A0028C2 00002868 000036E8
002810  00002890 0Q006FFF 00002800 00006FCC   FFFFFF34 00002858 000027F0 00004140
002840  00000000 00000000 00000000 00000000   C0000000 00000000 00008000 08000000
002870  00002974 00002938 02002974 20000050   47000000 47000000 00000000 00000000
0028A0  00002B48 08900909 000029C4 00000000   07004700 00000000 090029C4 20000050
0028D0  00002890 0A02D24F A102A152 5810A20E   58F10010 45EF000C 5810A212 58F10010
002900  A103A0B2 D213A115 A0B7D203 A133A0CB   D204A145 A0D45810 A20E58F1 001045EF
002930  A1A6A216 47F0A026 D24FA102 A1A8F321   A123A1A6 96F0A125 F363A143 A1A296F0
002960  4110A206 4500A0AE 00002858 00002890   0A020A0E F1F2F3F5 F6C2D6D3 E3404040
002990  F6404040 4040F5F2 52187F40 40404040   40404040 40404040 40404040 40404040
0029C0  40404040 40F1F2F3 F5F64040 40404040   40404040 404040C2 D6D3E340 40404040
0029F0  40404040 40F2F1F5 F6404040 40404040   40404040 404040F5 F2F1F8F7 40404040
002A20  40404040 40404040 40404040 C4C5E2C3   D9C9D7E3 C9D6D540 40404040 40404040
002A50  40404040 404040C1 D4D6E4D5 E3404040   40404040 0000000C 000CE3C8 C540D5E4
002A80  D6C3C5E2 E2C5C440 C9E24040 40404040   40404040 E3C8C540 E3D6E3C1 D340C1D4
002AB0  40404040 40404040 404007FE 615C0000   5B5BC2D6 D7C5D540 5B5BC2C3 D3D6E2C5
002AE0  0A320000 0A320000 47F0F01A 0A320000   C9D1C3C6 E9C9E9F3 F3F20A00 91801002
002B10  10044780 F04858E0 F0609140 10024780   F01A58E0 101C07FE D501F05C E0004770
002B40  BA0028F4 0A320000 0A320000 0A320000   0A320000 47F0F01A C9D1C4C6 E9E9E9E9
002B70  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000

006FE0  00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000
```

Figure A-35 Hexadecimal Core Dump

SUMMARY

The following chart summarizes the representation of digits, letters of the alphabet and special characters with EBCDIC.

THE EXTENDED BCD BYTE



SUMMARY

- A hexadecimal F (1111) in the zone portion of the byte indicates numeric data is stored in the byte in the zoned decimal format.
- A hexadecimal C (1100) is used in the zone portion of the byte when representing the letters of the alphabet A–I.
- A hexadecimal D (1101) is used in the zone portion of the byte when representing the letters of the alphabet J–R.
- A hexadecimal E (1110) is used in the zone portion of the byte when representing the letters of the alphabet S – Z.

BINARY NOTATION

Computers using the eight bit byte structure for internal data representation also have the ability to represent data in a pure binary format. Binary numbers are normally represented in a fixed length format consisting of a one bit sign followed by a series of bits or cores to represent the numeric value.

A one bit sign followed by a fifteen bit binary field is commonly called a HALFWORD. A Halfword requires two bytes or sixteen bits of storage.

General Format

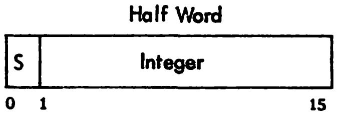


Figure A-36 General Form – Half Word

A one bit sign followed by a thirty-one bit binary field is called a WORD. A Word requires four bytes of storage or thirty-two bits.

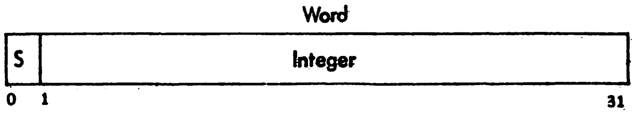


Figure A-37 General Form – Word

Positive numbers are represented in true binary notation with a sign bit of zero. The number +198 would be represented in a halfword as follows.

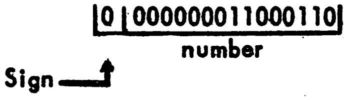


Figure A-38 Number 198 Represented in Binary

Negative numbers are represented in a two's complement notation with a "one" in the sign bit position. The two's complement of a binary number is obtained by inverting each bit of the number and adding a one in the low order bit position.

In COBOL data stored in a binary form is referred to as COMPUTATIONAL format.

APPENDIX B

COBOL Format Notation

Throughout this publication, basic formats are prescribed for various elements of COBOL. These generalized descriptions are intended to guide the programmer in writing his own statements. They are presented in a uniform system of notation, explained in the following paragraphs. Although it is not part of COBOL, this notation is useful in describing COBOL.

1. All words printed entirely in capital letters are reserved words. These are words that have preassigned meanings in COBOL. In all formats, words in capital letters represent an actual occurrence of those words. If any such word is incorrectly spelled, it will not be recognized as a reserved word and may cause an error in the program.

2. All underlined reserved words are required unless the portion of the format containing them is itself optional. These are key words. If any such word is missing or is incorrectly spelled, it is considered an error in the program. Reserved words not underlined may be included or omitted at the option of the programmer. These words are used only for the sake of readability; they are called optional words and, when used, must be correctly spelled.

3. The characters +, -, <, >, =, when appearing in formats, although not underlined, are required when such formats are used.

4. All punctuation and other special characters (except those symbols cited in the following paragraphs) represent the actual occurrence of those characters. Punctuation is essential where it is shown. Additional punctuation can be inserted, according to the rules for punctuation specified in this publication.

5. Words that are printed in lower-case letters represent information to be supplied by the programmer. All such words are defined in the accompanying text.

6. In order to facilitate references to them in text, some lower-case words are followed by a hyphen and a digit or letter. This modification does not change the syntactical definition of the word.

7. Certain entries in the formats consist of a capitalized word(s) followed by the word "Clause" or "Statement." These designate clauses or statements that are described in other formats, in appropriate sections of the text.

8. Square brackets ([ ]) are used to indicate that the enclosed item may be used or omitted, depending on the requirements of the particular program. When two or more items are stacked within brackets, one or none of them may occur.

9. Braces ({ }) enclosing vertically stacked items indicate that one of the enclosed items is obligatory.

10. The ellipsis (...) indicates that the immediately preceding unit may occur once, or any number of times in succession. A unit means either a single lower-case word, or a group of lower-case words and one or more reserved words enclosed in brackets or braces. If a term is enclosed in brackets or braces, the entire unit of which it is a part must be repeated when repetition is specified.

IDENTIFICATION DIVISION — BASIC FORMATS

$\left\{\begin{array}{l}\text{IDENTIFICATION DIVISION.}\\ \text{ID DIVISION.}\end{array}\right\}$

PROGRAM-ID. *program-name.*

AUTHOR. [*comment-entry*] . . .

INSTALLATION. [*comment-entry*] . . .

DATE-WRITTEN. [*comment-entry*] . . .

DATE-COMPILED. [*comment-entry*] . . .

SECURITY. [*comment-entry*] . . .

REMARKS. [*comment-entry*] . . .

It will be noted in the formats shown above and in subsequent format notations that some of the entries are shaded. These shaded portions of the formats depict additions which IBM has made to the basic ANSI COBOL Compiler and for those compilers executed on an IBM machine, these entries will be applicable. On other compilers, they may or may not be applicable.

ENVIRONMENT DIVISION — BASIC FORMATS

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. computer-name.
                                                   ⎧ WORDS      ⎫
OBJECT-COMPUTER. computer-name [MEMORY SIZE integer ⎨ CHARACTERS ⎬ ].
                                                   ⎩ MODULES    ⎭
SPECIAL-NAMES. [function-name-1 IS mnemonic-name] . . .

  [function-name-2 [IS mnemonic-name]

        {ON STATUS IS condition-name-1 [OFF STATUS IS condition-name-2]}
        {OFF STATUS IS condition-name-2 [ON STATUS IS condition-name-1]} ] . . .

  [CURRENCY SIGN IS literal]

  [DECIMAL-POINT IS COMMA].

INPUT-OUTPUT SECTION.

FILE-CONTROL.

  {SELECT [OPTIONAL] file-name

  ASSIGN TO [integer] system-name-1 [system-name-2] . . .

        [FOR MULTIPLE {REEL} ]
                      {UNIT}

  RESERVE {NO     } ALTERNATE [AREA ]
          {integer}           [AREAS]

  {FILE-LIMIT IS   } {data-name-1} THRU {data-name-2}
  {FILE-LIMITS ARE } {literal-1  }      {literal-2  }

        [ {data-name-3} THRU {data-name-4} ] . . .
          {literal-3  }      {literal-4  }

  ACCESS MODE IS {SEQUENTIAL}
                 {RANDOM    }

  PROCESSING MODE IS SEQUENTIAL

  ACTUAL KEY IS data-name

  NOMINAL KEY IS data-name

  RECORD KEY IS data-name

  TRACK-AREA IS integer CHARACTERS } . . .

I-O-CONTROL.

  RERUN ON system-name EVERY integer RECORDS OF file-name

  SAME [RECORD] AREA FOR file-name-1 {file-name-2} . . .

  MULTIPLE FILE TAPE CONTAINS file-name-1 [POSITION integer-1]

        [file-name-2 [POSITION integer-2]] . . .

  APPLY WRITE-ONLY ON file-name-1 [file-name-2] . . .

  APPLY EXTENDED-SEARCH ON file-name-1 [file-name-2] . . .

  APPLY WRITE-VERIFY ON file-name-1 [file-name-2] . . .

  APPLY CYL-OVERFLOW OF integer TRACKS ON file-name-1 [file-name-2] . . .

  APPLY {MASTER-INDEX} TO device-number ON file-name-1 [file-name-2] . . .
        {CYL-INDEX   }

  APPLY CORE-INDEX TO data-name ON file-name-1 [file-name-2] . . .
```

DATA DIVISION – BASIC FORMATS

```
DATA DIVISION.

FILE SECTION.

FD file-name

   BLOCK CONTAINS [integer-1 TO] integer-2 {CHARACTERS}
                                           {RECORDS   }

   RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS

   RECORDING MODE IS mode

          {RECORD IS  }  {OMITTED                             }
   LABEL  {RECORDS ARE}  {STANDARD                            }
                         {data-name-1 [data-name-2] . . .     }

   VALUE OF data-name-1 IS {data-name-2} [data-name-3 IS {data-name-4}  ] . . .
                           {literal-1  }                 {literal-2  }

          {RECORD IS  }
   DATA   {RECORDS ARE} data-name-1 [data-name-2] . . . .


01-49  {data-name-1}
       {FILLER     }

   REDEFINES data-name-2

   BLANK WHEN ZERO

   {JUSTIFIED} RIGHT
   {JUST     }

   {PICTURE} IS character string
   {PIC    }

   {SYNCHRONIZED} [LEFT ]
   {SYNC        } [RIGHT]

                 { INDEX               }
                 { DISPLAY             }
                 { {COMPUTATIONAL  }   }
                 { {COMP           }   }
                 { {COMPUTATIONAL-1}   }
   [USAGE IS]    { {COMP-1         }   }
                 { {COMPUTATIONAL-2}   }
                 { {COMP-2         }   }
                 { {COMPUTATIONAL-3}   }
                 { {COMP-3         }   }
                 { DISPLAY-ST          }

88  condition-name {VALUE IS  } literal-1 [THRU literal-2]
                   {VALUES ARE}

          [literal-3] [THRU literal-4]] . . . .

66  data-name-1 RENAMES data-name-2 [THRU data-name-3] .
```

NOTE: Formats of the OCCURS Clause are included with TABLE HANDLING formats.

```
WORKING-STORAGE SECTION.
77  data-name-1

01-49 {data-name-1}
      {FILLER     }

  REDEFINES data-name-2
  BLANK WHEN ZERO
  {JUSTIFIED}
  {JUST     } RIGHT
  {PICTURE}
  {PIC    } IS character string
  {SYNCHRONIZED} [LEFT ]
  {SYNC        } [RIGHT]
                 {  INDEX                }
                 {  DISPLAY              }
                 { {COMPUTATIONAL  }     }
                 { {COMP           }     }
                 { {COMPUTATIONAL-1}     }
  [USAGE IS]     { {COMP-1         }     }
                 { {COMPUTATIONAL-2}     }
                 { {COMP-2         }     }
                 { {COMPUTATIONAL-3}     }
                 { {COMP-3         }     }
                 {  DISPLAY-ST           }
  VALUE IS literal.

88  condition-name {VALUE  IS  } literal-1 [THRU literal-2]
                   {VALUES ARE }
          [literal-3] [THRU literal-4]] . . . .
66  data-name-1 RENAMES data-name-2 [THRU data-name-3] .
```

NOTE: Formats of the OCCURS Clause are included with TABLE HANDLING formats.

```
LINKAGE SECTION.

77  data-name-1

01-49  {data-name-1}
       {FILLER     }

   REDEFINES data-name-2

   BLANK WHEN ZERO

   {JUSTIFIED} RIGHT
   {JUST     }

   {PICTURE} IS character string
   {PIC    }

   {SYNCHRONIZED} [LEFT ]
   {SYNC        } [RIGHT]

                  { INDEX                }
                  { DISPLAY              }
                  {{COMPUTATIONAL  }     }
                  {{COMP           }     }
                  {{COMPUTATIONAL-1}     }
   [USAGE IS]     {{COMP-1         }     }
                  {{COMPUTATIONAL-2}     }
                  {{COMP-2         }     }
                  {{COMPUTATIONAL-3}     }
                  {{COMP-3         }     }
                  { DISPLAY-ST           }

88  condition-name {VALUE IS  } literal-1 [THRU literal-2]
                   {VALUES ARE}

         [literal-3] [THRU literal-4]] . . .

66  data-name-1 RENAMES data-name-2 [THRU data-name-3].
```

NOTE: Formats of the OCCURS Clause are included with TABLE HANDLING formats.

PROCEDURE DIVISION — BASIC FORMATS

```
{PROCEDURE DIVISION.                                        }
{PROCEDURE DIVISION USING identifier-1 [identifier-2] . . . }
```

ACCEPT Statement

```
                          {CONSOLE      }
ACCEPT identifier [FROM   {mnemonic-name}  ]
```

ADD Statement

FORMAT 1

```
    {identifier-1} [identifier-2]
ADD {literal-1   } [literal-2   ] . . . TO identifier-m [ROUNDED]

        [identifier-n [ROUNDED]] . . . [ON SIZE ERROR imperative-statement]
```

FORMAT 2

```
    {identifier-1} {identifier-2} [identifier-3]
ADD {literal-1   } {literal-2   } [literal-3   ] . . .

    GIVING identifier-m [ROUNDED] [ON SIZE ERROR imperative-statement]
```

FORMAT 3

```
    {CORRESPONDING}
ADD {CORR         } identifier-1 TO identifier-2 [ROUNDED]

    [ON SIZE ERROR imperative-statement]
```

ALTER Statement

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2

        [procedure-name-3 TO [PROCEED TO] procedure-name-4] . . .
```

CALL Statement

```
CALL literal-1 [USING identifier-1 [identifier-2] . . .]
```

CLOSE Statement

```
                  [REEL]        {NO REWIND}
CLOSE file-name-1 [UNIT] [WITH  {LOCK     } ]

                  [REEL]        {NO REWIND}
     [file-name-2 [UNIT] [WITH  {LOCK     } ]] . . .
```

COMPUTE Statement

```
                                     { arithmetic-expression }
COMPUTE identifier-1 [ROUNDED]  =    { identifier-2          }
                                     { literal-1             }

        [ON SIZE ERROR imperative-statement]
```

DECLARATIVE Section

$\underline{\text{PROCEDURE}}\ \underline{\text{DIVISION}}.$

$\underline{\text{DECLARATIVES}}.$

$\{\textit{section-name}\ \underline{\text{SECTION}}.\ \underline{\text{USE}}\ \textit{sentence.}$

$\{\textit{paragraph-name.}\ \{\textit{sentence}\} \ldots\} \ldots\} \ldots$

$\underline{\text{END}}\ \underline{\text{DECLARATIVES}}.$

DISPLAY Statement

$$\underline{\text{DISPLAY}} \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\end{array}\right\} \left[\begin{array}{l}\textit{identifier-2}\\ \textit{literal-2}\end{array}\right] \ldots \left[\underline{\text{UPON}} \left\{\begin{array}{l}\underline{\text{CONSOLE}}\\ \underline{\text{SYSPCH}}\\ \underline{\text{SYSPUNCH}}\\ \textit{mnemonic-name}\end{array}\right\}\right]$$

DIVIDE Statement

FORMAT 1

$$\underline{\text{DIVIDE}} \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\end{array}\right\} \underline{\text{INTO}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

FORMAT 2

$$\underline{\text{DIVIDE}} \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\end{array}\right\} \left\{\begin{array}{l}\underline{\text{INTO}}\\ \underline{\text{BY}}\end{array}\right\} \left\{\begin{array}{l}\textit{identifier-2}\\ \textit{literal-2}\end{array}\right\} \underline{\text{GIVING}}\ \textit{identifier-3}\ [\underline{\text{ROUNDED}}]$$

$$[\underline{\text{REMAINDER}}\ \textit{identifier-4}]\ [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

ENTER Statement

$\underline{\text{ENTER}}\ \textit{language-name}\ [\textit{routine-name}].$

ENTRY Statement

$\underline{\text{ENTRY}}\ \textit{literal}\ [\underline{\text{USING}}\ \textit{identifier-1}\ [\textit{identifier-2}] \ldots]$

EXAMINE Statement

FORMAT 1

$$\underline{\text{EXAMINE}}\ \textit{identifier}\ \underline{\text{TALLYING}} \left\{\begin{array}{l}\underline{\text{UNTIL}}\ \underline{\text{FIRST}}\\ \underline{\text{ALL}}\\ \underline{\text{LEADING}}\end{array}\right\} \textit{literal-1}$$

$$[\underline{\text{REPLACING}}\ \underline{\text{BY}}\ \textit{literal-2}]$$

FORMAT 2

$$\underline{\text{EXAMINE}}\ \textit{identifier}\ \underline{\text{REPLACING}} \left\{\begin{array}{l}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \underline{\text{FIRST}}\\ \underline{\text{UNTIL}}\ \underline{\text{FIRST}}\end{array}\right\} \textit{literal-1}\ \underline{\text{BY}}\ \textit{literal-2}$$

EXIT Statement

$\textit{paragraph-name.}\ \underline{\text{EXIT}}\ [\underline{\text{PROGRAM}}].$

GOBACK Statement

$\underline{\text{GOBACK}}.$

GO TO Statement

FORMAT 1

$$\underline{\text{GO}}\ \underline{\text{TO}}\ \textit{procedure-name-1}$$

FORMAT 2

$$\underline{\text{GO}}\ \underline{\text{TO}}\ \textit{procedure-name-1}\ [\textit{procedure-name-2}]\ \ldots\ \underline{\text{DEPENDING}}\ \text{ON}\ \textit{identifier}$$

FORMAT 3

$$\underline{\text{GO}}\ \underline{\text{TO}}.$$

IF Statement

$$\underline{\text{IF}}\ \textit{condition}\ \text{THEN}\ \left\{\begin{matrix}\underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\\ \textit{statement-1}\end{matrix}\right\}\ \left\{\begin{matrix}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{matrix}\right\}\ \left\{\begin{matrix}\underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\\ \textit{statement-2}\end{matrix}\right\}$$

MOVE Statement

FORMAT 1

$$\underline{\text{MOVE}}\ \left\{\begin{matrix}\textit{identifier-1}\\ \textit{literal}\end{matrix}\right\}\ \underline{\text{TO}}\ \textit{identifier-2}\ [\textit{identifier-3}]\ \ldots$$

FORMAT 2

$$\underline{\text{MOVE}}\ \left\{\begin{matrix}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{matrix}\right\}\ \textit{identifier-1}\ \underline{\text{TO}}\ \textit{identifier-2}$$

MULTIPLY Statement

FORMAT 1

$$\underline{\text{MULTIPLY}}\ \left\{\begin{matrix}\textit{identifier-1}\\ \textit{literal-1}\end{matrix}\right\}\ \underline{\text{BY}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$
$$[\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

FORMAT 2

$$\underline{\text{MULTIPLY}}\ \left\{\begin{matrix}\textit{identifier-1}\\ \textit{literal-1}\end{matrix}\right\}\ \underline{\text{BY}}\ \left\{\begin{matrix}\textit{identifier-2}\\ \textit{literal-2}\end{matrix}\right\}\ \underline{\text{GIVING}}\ \textit{identifier-3}$$
$$[\underline{\text{ROUNDED}}]\ [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

NOTE Statement

$$\underline{\text{NOTE}}\ \textit{character string}$$

OPEN Statement

$$\underline{\text{OPEN}}\ [\underline{\text{INPUT}}\ \{\textit{file-name}\ \left[\begin{matrix}\underline{\text{REVERSED}}\\ \text{WITH}\ \underline{\text{NO}}\ \underline{\text{REWIND}}\end{matrix}\right]\}\ \ldots]$$
$$[\underline{\text{OUTPUT}}\ \{\textit{file-name}\ [\text{WITH}\ \underline{\text{NO}}\ \underline{\text{REWIND}}]\}\ \ldots]$$
$$[\underline{\text{I-O}}\ \{\textit{file-name}\}\ \ldots]$$

PERFORM Statement

FORMAT 1

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}]$$

FORMAT 2

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}]\ \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{integer-1}\end{array}\right\}\ \underline{\text{TIMES}}$$

FORMAT 3

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}]\ \underline{\text{UNTIL}}\ \textit{condition-1}$$

FORMAT 4

$$\underline{\text{PERFORM}}\ \textit{procedure-name-1}\ [\underline{\text{THRU}}\ \textit{procedure-name-2}]$$

$$\underline{\text{VARYING}}\ \left\{\begin{array}{l}\textit{index-name-1}\\ \textit{identifier-1}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\textit{index-name-2}\\ \textit{literal-2}\\ \textit{identifier-2}\end{array}\right\}\ \underline{\text{BY}}$$

$$\left\{\begin{array}{l}\textit{literal-3}\\ \textit{identifier-3}\end{array}\right\}\ \underline{\text{UNTIL}}\ \textit{condition-1}$$

$$[\underline{\text{AFTER}}\ \left\{\begin{array}{l}\textit{index-name-4}\\ \textit{identifier-4}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\textit{index-name-5}\\ \textit{literal-5}\\ \textit{identifier-5}\end{array}\right\}\ \underline{\text{BY}}$$

$$\left\{\begin{array}{l}\textit{literal-6}\\ \textit{identifier-6}\end{array}\right\}\ \underline{\text{UNTIL}}\ \textit{condition-2}$$

$$[\underline{\text{AFTER}}\ \left\{\begin{array}{l}\textit{index-name-7}\\ \textit{identifier-7}\end{array}\right\}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\textit{index-name-8}\\ \textit{literal-8}\\ \textit{identifier-8}\end{array}\right\}\ \underline{\text{BY}}$$

$$\left\{\begin{array}{l}\textit{literal-9}\\ \textit{identifier-9}\end{array}\right\}\ \underline{\text{UNTIL}}\ \textit{condition-3}]]$$

READ Statement

$$\underline{\text{READ}}\ \textit{file-name}\ \text{RECORD}\ [\underline{\text{INTO}}\ \textit{identifier}]\ \left\{\begin{array}{l}\text{AT }\underline{\text{END}}\\ \underline{\text{INVALID}}\ \text{KEY}\end{array}\right\}\ \textit{imperative-statement}$$

REWRITE Statement

$$\underline{\text{REWRITE}}\ \textit{record-name}\ [\underline{\text{FROM}}\ \textit{identifier}]\ \underline{\text{INVALID}}\ \text{KEY}\ \textit{imperative-statement}$$

SEEK Statement

$$\underline{\text{SEEK}}\ \textit{file-name}\ \text{RECORD}$$

START Statement

$$\underline{\text{START}}\ \textit{file-name}\ \underline{\text{INVALID KEY}}\ \textit{imperative-statement}$$

STOP Statement

$$\underline{\text{STOP}}\ \left\{\begin{array}{l}\underline{\text{RUN}}\\ \textit{literal}\end{array}\right\}$$

SUBTRACT Statement

FORMAT 1

$$\underline{\text{SUBTRACT}}\ \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\end{array}\right\}\ \left[\begin{array}{l}\textit{identifier-2}\\ \textit{literal-2}\end{array}\right] \ldots\ \underline{\text{FROM}}\ \textit{identifier-m}\ [\underline{\text{ROUNDED}}]$$

$$[\textit{identifier-n}\ [\underline{\text{ROUNDED}}]]\ \ldots\ [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

FORMAT 2

$$\underline{\text{SUBTRACT}}\ \left\{\begin{array}{l}\textit{identifier-1}\\ \textit{literal-1}\end{array}\right\}\ \left[\begin{array}{l}\textit{identifier-2}\\ \textit{literal-2}\end{array}\right] \ldots\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\textit{identifier-m}\\ \textit{literal-m}\end{array}\right\}\ \underline{\text{GIVING}}$$

$$\textit{identifier-n}\ [\underline{\text{ROUNDED}}]\ [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

FORMAT 3

$$\underline{\text{SUBTRACT}}\ \left\{\begin{array}{l}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{array}\right\}\ \textit{identifier-1}\ \underline{\text{FROM}}\ \textit{identifier-2}\ [\underline{\text{ROUNDED}}]$$

$$[\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \textit{imperative-statement}]$$

TRANSFORM Statement

$$\underline{\text{TRANSFORM}}\ \textit{identifier-3}\ \text{CHARACTERS}\ \underline{\text{FROM}}\ \left\{\begin{array}{l}\textit{figurative-constant-1}\\ \textit{nonnumeric-literal-1}\\ \textit{identifier-1}\end{array}\right\}$$

$$\underline{\text{TO}}\ \left\{\begin{array}{l}\textit{figurative-constant-2}\\ \textit{nonnumeric-literal-2}\\ \textit{identifier-2}\end{array}\right\}$$

USE Sentence

FORMAT 1

Option 1:

USE {BEFORE / AFTER} STANDARD [BEGINNING] [REEL / FILE / UNIT]

LABEL PROCEDURE ON { {*file-name*} . . . / OUTPUT / INPUT / I-O }

Option 2:

USE {BEFORE / AFTER} STANDARD [ENDING] [REEL / FILE / UNIT]

LABEL PROCEDURE ON { {*file-name*} . . . / OUTPUT / INPUT / I-O }.

FORMAT 2

USE AFTER STANDARD ERROR PROCEDURE

ON { {*file-name-1*} / *file-name-2* GIVING *data-name-1* [*data-name-2*] / INPUT / OUTPUT / I-O }.

WRITE Statement

FORMAT 1

WRITE *record-name* [FROM *identifier-1*]

[{BEFORE / AFTER} ADVANCING { *identifier-2* LINES / *integer* LINES / *mnemonic-name* }]

[AT {END-OF-PAGE / EOP} *imperative-statement*]

FORMAT 2

WRITE *record-name* [FROM *identifier-1*]

AFTER POSITIONING {*identifier-2* / *integer*} LINES

[AT {END-OF-PAGE / EOP} *imperative-statement*]

FORMAT 3

WRITE *record-name* [FROM *identifier-1*] INVALID KEY *imperative-statement*

TABLE HANDLING — BASIC FORMATS

Data Division Table Handling Formats

OCCURS Clause

Format 1

OCCURS *integer* TIMES

[INDEXED BY *index-name-1* [*index-name-2*] . . .]

Format 2

OCCURS *integer* TIMES [DEPENDING ON *data-name*]

[INDEXED BY *index-name-1* [*index-name-2*] . . .]

USAGE IS INDEX Clause

[USAGE IS] INDEX

Procedure Division Table Handling Formats

SET Statement

Format 1

$$\underline{\text{SET}} \begin{Bmatrix} \textit{index-name-1} & [\textit{index-name-2}] \ldots \\ \textit{identifier-1} & [\textit{identifier-2}\ \ ] \ldots \end{Bmatrix} \underline{\text{TO}} \begin{Bmatrix} \textit{index-name-3} \\ \textit{identifier-3} \\ \textit{literal-1} \end{Bmatrix}$$

Format 2

$$\underline{\text{SET}}\ \textit{index-name-4}\ [\textit{index-name-5}] \ldots \begin{Bmatrix} \underline{\text{UP}}\ \underline{\text{BY}} \\ \underline{\text{DOWN}}\ \underline{\text{BY}} \end{Bmatrix} \begin{Bmatrix} \textit{identifier-4} \\ \textit{literal-2} \end{Bmatrix}$$

SEGMENTATION — BASIC FORMATS

Priority Numbers in Procedure Division

section-name SECTION [*priority-number*].

SOURCE PROGRAM LIBRARY FACILITY

COPY Statement

COPY *library name* [SUPPRESS].

DEBUGGING LANGUAGE — BASIC FORMATS

Procedure Division Debugging Formats

EXHIBIT Statement

$$\underline{\text{EXHIBIT}} \left\{ \begin{array}{l} \underline{\text{NAMED}} \\ \underline{\text{CHANGED}}\ \underline{\text{NAMED}} \\ \underline{\text{CHANGED}} \end{array} \right\} \left\{ \begin{array}{l} \textit{identifier-1} \\ \textit{nonnumeric-literal-1} \end{array} \right\} \left[\begin{array}{l} \textit{identifier-2} \\ \textit{nonnumeric-literal-2} \end{array} \right] \ldots$$

ON (Count-conditional) Statement

$$\underline{\text{ON}}\ \textit{integer-1}\ [\underline{\text{AND}}\ \underline{\text{EVERY}}\ \textit{integer-2}]\ [\underline{\text{UNTIL}}\ \textit{integer-3}]$$

$$\left\{ \begin{array}{l} \textit{imperative-statement} \ldots \\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{ELSE}} \\ \underline{\text{OTHERWISE}} \end{array} \right\} \left\{ \begin{array}{l} \textit{statement} \ldots \\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}} \end{array} \right\}$$

READY/RESET TRACE Statement

$$\left\{ \begin{array}{l} \underline{\text{READY}} \\ \underline{\text{RESET}} \end{array} \right\} \underline{\text{TRACE}}$$

Compile-Time Debugging Packet

DEBUG Card

$$\underline{\text{DEBUG}} \quad \textit{location}$$

FORMAT CONTROL — BASIC FORMATS

EJECT Statement

1	Area B
	$\underline{\text{EJECT}}$

SKIP1, SKIP2, SKIP3 Statements

1	Area B
	$\left\{ \begin{array}{l} \underline{\text{SKIP1}} \\ \underline{\text{SKIP2}} \\ \underline{\text{SKIP3}} \end{array} \right\}$

APPENDIX C

COBOL

REFERENCE SUMMARY

CHARACTER SET

The complete COBOL character set consists of the following 51 characters:

Digits 0 through 9
Letters A through Z
Special characters:

	Blank or space
+	Plus sign
-	Minus sign or hyphen
*	Check protection symbol, asterisk
/	Slash
=	Equal sign
>	Inequality sign (greater than)
<	Inequality sign (less than)
$	Dollar sign
,	Comma
.	Period or decimal point
'	Quotation mark
(	Left parenthesis
)	Right parenthesis
;	Semicolon

Of the previous set, the following characters are used for words:

0 through 9
A through Z
- (hyphen)

The following characters are used for punctuation:

'	Quotation mark
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Period
;	Semicolon

The following characters are used in arithmetic expressions:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The following characters are used in relation tests:

>	Greater than
<	Less than
=	Equal to

All of the preceding characters are contained in the COBOL character set. In addition, the programmer can use, as characters in non-numeric literals, any characters (except the quotation mark) included in the IBM Extended Binary-Coded-Decimal Interchange Code; however, such characters may be unacceptable to COBOL for other computers.

WORD FORMATION

A word is composed of a combination of not more than 30 characters, chosen from the following set of 37 characters:

0 through 9 (digits)
A through Z (letters)
- (hyphen)

A word must not begin or end with a hyphen. A word is ended by a space, or by proper punctuation. Embedded hyphens are permitted. All words in COBOL are either reserved words, which have preassigned meanings in COBOL, or programmer-supplied names. Each type of name is discussed in the section of this publication in which it is first mentioned.

TYPES OF NAMES

There are three types of names used in a COBOL program:

1. A data-name is a word that contains at least one alphabetic character and identifies a data item in the Data Division. The following are formed according to the rules for data-names:

   file-names
   sort-file-description-names
   record-names
   report-names

2. A condition-name is a name given to a specific value, set of values, or range of values, within the complete set of values that a particular data item may assume. The data item itself is called a conditional variable. The condition-name must contain at least one alphabetic character (see "Data Division" and the discussion of "Special-Names" in "Environment Division").

3. A procedure-name is either a paragraph-name or a section-name. A procedure-name may be composed solely of numeric characters. Two numeric procedure-names are equivalent if, and only if, they are composed of the same number of digits and have the same value (see "Procedure Division"). The following are formed according to the rules for procedure-names:

   library-names
   program-names

QUALIFICATION OF NAMES

Every name used in a COBOL source program must be unique within the source program, either because no other name has the identical spelling, or because the name exists within a hierarchy of names (so that the name can be made unique by mentioning one or more of the higher levels of the hierarchy). The higher levels are called qualifiers when used in this way, and the process is called qualification.

The following rules apply to the qualification of names:

1. The word OF or IN must precede each qualifying name, and the names must appear in ascending order of hierarchy.
2. A qualifier must be of a higher level and within the same hierarchy as the name it is qualifying.
3. The same name must not appear at two levels in a hierarchy in such a manner that it would appear to qualify itself.
4. The highest level qualifier must be unique. Each qualifying name must be unique at its own level within the hierarchy of the immediately higher qualifier.
5. Qualification when not needed is permitted.
6. Qualifiers must not be subscripted, although the entire qualified name may be subscripted.
7. The total number of characters x cannot exceed 300 where:
   x = T + 4N
   T is the number of characters in all names, and
   N is the number of data names, including their qualifiers.
8. Regardless of qualification, procedure names and data names must not be the same.

COBOL RESERVED WORDS

The following is a list of reserved words. It should be noted that all of the words in the list are not reserved words for all COBOL compilers. It is the best policy, however, to not use any of the words in the list as data-names or procedure-names. This will eliminate any possibility of using reserved words on any COBOL compiler.

ACCEPT
ACCESS
ACTUAL
ADD
~~ADDRESS~~
ADVANCING
~~AFTER~~
~~ALL~~
ALPHABETIC
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BASIS
BEFORE
BEGINNING
BLANK
BLOCK
BY

CALL
CANCEL
CF
CH
CHANGED
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COM-REG
COMMA
COMP
COMP-1
COMP-2
COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONSTANT
CONTAINS
CONTROL
CONTROLS
COPY
CORE-INDEX
CORR
CORRESPONDING
CSP
CURRENCY
CURRENT-DATE
CYL-INDEX
CYL-OVERFLOW
C01
C02
C03
C04
C05
C06
C07
C08
C09
C10
C11
C12

DATA
DATE-COMPILED
DATE-WRITTEN
DE
DEBUG
DECIMAL-POINT
DECLARATIVES
~~DELETE~~
DEPENDING
DESCENDING
~~DETAIL~~
DISP
~~DISPLAY~~
DISPLAY-ST
DISPLAY-n
DIVIDE
DIVISION
DOWN

EJECT
ELSE
END
END-OF-PAGE
ENDING
ENTER
ENTRY
ENVIRONMENT
EOP
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXCEEDS
EXHIBIT
EXIT
EXTENDED-SEARCH

FD
~~FILE~~
~~FILE-CONTROL~~
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

~~GENERATE~~
~~GIVING~~
GO
GOBACK
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
~~HOLD~~
I-O
I-O-CONTROL
ID
IDENTIFICATION
IF
IN
INDEX
INDEX-n
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSERT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY
KEYS

LABEL
LABEL-RETURN
LAST
LEADING
LEAVE
LEFT
LESS
LIBRARY
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES
LOWER-BOUND
LOWER-BOUNDS

MASTER-INDEX
MEMORY
MODE
MODULES
MORE-LABELS
MOVE
~~MULTIPLE~~
MULTIPLY

NAMED
NEGATIVE
NEXT
NO
NOMINAL
NOT
NOTE
NSTD-REELS
NUMBER
NUMERIC
~~OBJECT-COMPUTER~~
OBJECT-PROGRAM
OCCURS
OF
OFF
OH
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT
OV
OVERFLOW

PAGE
~~PAGE-COUNTER~~
PERFORM
PF
PH
PIC
PICTURE
PLUS
POSITION
POSITIONING
POSITIVE
PREPARED
PRINT-SWITCH
PRIORITY
PROCEDURE
PROCEED
PROCESS
PROCESSING
PROGRAM
PROGRAM-ID

QUOTE
QUOTES

RANDOM
RANGE
RD
READ
READY
RECORD
RECORD-OVERFLOW
RECORDING
RECORDS
REDEFINES
REEL
RELEASE
REMAINDER
REMARKS
RENAMES
REORG-CRITERIA
REPLACING
REPORT
REPORTING
REPORTS
REREAD
RERUN
RESERVE
RESET
RETURN
RETURN-CODE
REVERSED
REWIND
REWRITE
RF
RH
~~RIGHT~~
ROUNDED
RUN
SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SELECTED
SENTENCE
SEQUENTIAL
SET
SIGN
SIZE
SKIP1
SKIP2
SKIP3
SORT
SORT-CORE-SIZE
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
~~SPECIAL-NAMES~~
STANDARD
START
STATUS
STOP
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SUSPEND
SYNC
SYNCHRONIZED
SYSIN
SYSIPT
SYSLST
SYSOUT
SYSPCH
SYSPUNCH
S01
S02

TALLY
TALLYING
TAPE
TERMINATE
THAN
THEN
THROUGH
THRU
TIME-OF-DAY
TIMES
TO
TOTALED
TOTALING
TRACE
TRACK
TRACK-AREA
TRACK-LIMIT
TRACKS
TRANSFORM
TYPE
UNEQUAL
UNIT
UNTIL
UP
UPON
UPPER-BOUND
UPPER-BOUNDS
UPSI-0
UPSI-1
UPSI-2
UPSI-3
UPSI-4
UPSI-5
UPSI-6
UPSI-7
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
~~WORKING-STORAGE~~
WRITE
WRITE-ONLY
WRITE-VERIFY

ZERO
ZEROES
ZEROS

A literal is a constant that is not identified by a data-name in a program, but is completely defined by its own identity. A literal is either non-numeric (alphabetic or alphanumeric), numeric, or floating-point.

Non-Numeric Literals

A non-numeric literal must be bounded by quotation marks and may consist of any combination of characters in the IBM EBCDIC set, except quotation marks. All spaces enclosed by the quotation marks are included as part of the literal. A non-numeric literal may not exceed 120 characters in length.

The following are examples of non-numeric literals:

```
'EXAMINE CLOCK NUMBER'
'12565'
'PAGE 144 MISSING'
```

Numeric Literals

A numeric literal must contain at least one and not more than 18 digits. A numeric literal may consist of the characters 0 through 9, the plus sign or the minus sign, and the decimal point. It may contain only one sign character and only one decimal point. The sign, if present, must appear as the leftmost character in the numeric literal. If a numeric literal is unsigned, it is assumed to be positive.

A decimal point may appear anywhere within the numeric literal, except as the rightmost character. If a numeric literal does not contain a decimal point, it is considered to be a whole number.

The following are examples of numeric literals:

```
        1506798
        +12572.6
        -256.75
        .16
```

FIGURATIVE CONSTANTS

Figurative constants are a special type of literal. They are values that have been assigned fixed data-names. Figurative constants must not be bounded by quotation marks.

ZERO may be used in many places in a program as a numeric literal. It may not, however, be used in an arithmetic statement. The use of ZERO as a non-numeric literal is permitted. All other figurative constants are considered non-numeric. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

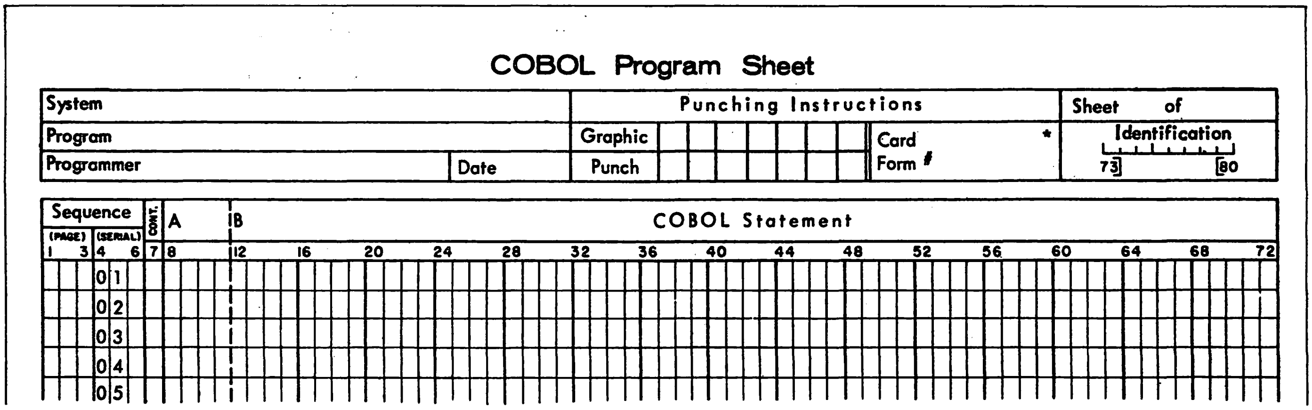
ZERO ZEROS ZEROES	Represents one or more zeros.
SPACE SPACES	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Represents one or more appearances of the highest value in the computer's collating sequence. (Hexadecimal 'FF')
LOW-VALUE LOW-VALUES	Represents one or more appearances of the lowest value in the computer's collating sequence. (Hexadecimal '00')
ALL 'character'	Represents one or more occurrences of the single character bounded by quotation marks. Character may not be a quotation mark.
QUOTE QUOTES	Represents the character '. Note that the use of the word QUOTE to represent the character ' at object time is not equivalent to the use of the symbol ' to bound a non-numeric literal.

When a figurative constant is used in such a way that the exact number of characters required cannot be determined, only one character is generated. For example, the statement DISPLAY ZEROES would produce one zero character since, in this case, the length of the sequence of zeros to be displayed cannot be determined.

COBOL PROGRAM SHEET

The purpose of the program sheet is to provide a standard way of writing COBOL source programs.

The Identification, Environment, Data, and Procedure Divisions which constitute a COBOL source program are written in the stated order. This program sheet, despite its necessary restrictions, is of a relatively free form. The programmer should note, however, that the rules for using it are precise and must be followed exactly. These rules take precedence over any other rules, with respect to spacing.



COBOL Program Sheet

System	Punching Instructions		Sheet of
Program	Graphic	Card Form # *	Identification 73 80
Programmer Date	Punch		

Sequence (PAGE) 1 3	(SERIAL) 4 6	CONT. 7	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 COBOL Statement
	01			
	02			
	03			
	04			
	05			

SEQUENCE NUMBER: (COLUMNS 1-6)

The sequence number must consist only of digits; letters and special characters should not be used. The sequence number has no effect on the source program and need not be written. If the programmer supplies sequence numbers in each program card, the compiler will check the source program cards and will indicate any errors in their sequence. If these columns are blank, no sequence error will be indicated.

CONTINUATION INDICATOR: (COLUMN 7)

When a non-numeric literal is of a length such that it cannot be contained on one line of a coding sheet, the following rules apply:

1. On every line containing a portion of a literal to be continued, the portion of the literal that is to be continued must not be terminated with a quotation mark.
2. On every line containing a portion of a literal being continued, the portion of the literal being continued must be immediately preceded by a quotation mark. This quotation mark may appear anywhere in Margin B, and may not be preceded by anything but spaces.
3. A hyphen must be punched in column 7 of each line in which the literal is being continued.

SOURCE PROGRAM STATEMENTS: (COLUMNS 8-72)

These columns are used for writing the COBOL source program.

PROGRAM IDENTIFICATION CODE: (COLUMNS 73-80)

These columns can be used to identify the program. Any character from the COBOL character set may be used, including the blank. The program identification code has no effect on the object program or the compiler.

MARGIN RESTRICTIONS

AREA A AND AREA B

Area A, columns 8 through 11, is reserved for the beginning of division headers, section-names, paragraph-names, level indicators, and certain level numbers. Area B occupies columns 12 through 72.

The division header must be the first line in a division. The division header starts in Area A with the division-name, followed by a space and the word DIVISION, and a period.

The name of a section starts in Area A of any line following the division header. The section-name is followed by a space, the word SECTION, and a period. If program segmentation is desired, a space and a priority number may follow the word SECTION. No other text may appear on the same line as the section-header, except USE and COPY sentences.

The name of a paragraph starts in Area A of any line following the division header. It is followed by a period followed by a space.

A paragraph consists of one or more successive sentences. The first sentence in a paragraph begins anywhere in Area B of either the same line as paragraph-name or the immediately following line. Each successive line in the paragraph starts anywhere in Area B.

PUNCTUATION

The following general rules of punctuation apply in writing COBOL source programs:

1. When any punctuation mark is indicated in a format in this publication, it is required.
2. A period, semicolon or comma, when used, must not be preceded by a space, but must be followed by a space.
3. A left parenthesis must not be followed immediately by a space; a right parenthesis must not be preceded immediately by a space.
4. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except in non-numeric literals.
5. When an arithmetic operator or an equal sign is used, it must be preceded by a space and followed by another space.
6. When the period or comma, or arithmetic operator characters are used in the PICTURE clause as editing characters, they are governed by rules for report items only.
7. A comma may be used as a separator between successive operands of a statement. A comma or semicolon may be used to separate a series of clauses , and a semicolon or the word THEN may be used to separate a series of statements.

TYPES OF DATA ITEMS

Several types of data items can be described in a COBOL source program. These data items are described in the following text. The format of the Record Description entry used to describe each of these items appears under the discussion of Record Description entries.

Alphabetic Item

An alphabetic item may contain any combination of the characters A through Z and space. Each alphabetic character is stored in a separate byte.

Alphanumeric Item

An alphanumeric item consists of any combination of characters in the IBM EBCDIC set. Each alphanumeric character is stored in a separate byte.

Report Item

A report item is an alphanumeric item containing only digits and/or special editing characters. It must not exceed 127 characters in length. A report item can be used only as a receiving field for numeric data. Each report character is stored in a separate byte (see PICTURE and BLANK WHEN ZERO clauses), except P and V which occupy no storage, and CR and DB which occupy two bytes each.

Elementary Items

An elementary item is a data item containing no subordinate items. For example, an 03 level data item followed immediately by another 03 item is an elementary item. Elementary items always have the picture clause specified.

Group Items

A group item is defined as a field having further subdivisions, so that it contains one or more elementary items. In addition, a group item may contain other groups. An item is a group item if, and only if, its level number is less than the level number of the immediately succeeding item, unless the succeeding level is 88. If an item is not a group item, then it is an elementary item, or, in the case of level 88, it is a condition-name.

The maximum length for any elementary or group item in a tape system is 4092 bytes. For a disk file, maximum length depends upon the particular disk model; 3625, 4096, and 2000 bytes for the IBM 2311, 2314, and 2321 direct-access devices respectively. An exception to these stated lengths is the use of a fixed-length Working-Storage group item, which may be as long as 32,767 bytes. If an IBM 2314 direct-access device is available, it should be used since it offers the advantage of longer track records.

Group items never have a picture associated with them.

Fixed-Point Items

Fixed-point items may be defined as external decimal, internal decimal, or binary. External decimal corresponds to the form in which information is represented initially for card input, or finally for printed or punched output. Such items may be converted (by moving) to the internal machine formats described as internal decimal or binary. Except when an item is a single digit in length, these formats require less storage than the external decimal format and can be used to save space on volumes. The binary mode of representation is particularly efficient for data-names used as subscripts. Computational results are the same, regardless of the particular format selected provided the intermediate computational results do not require more than 18 digit positions.

EXTERNAL DECIMAL ITEM: Decimal numbers in the System/360 zoned format are external decimal items.Each digit of a number is represented by a single byte, with the four low-order bits of each eight-bit byte containing the value of a digit. The four high-order bits of each byte are zone bits; the zone bits of the low-order byte represent the sign of the item. The maximum length of an external decimal item is 18 digits. For items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration interpreted as positive but which does not represent an overpunch.

INTERNAL DECIMAL ITEM: An internal decimal item consists of numeric characters 0 through 9 plus a sign, and represents a value not exceeding 18 digits in length. It appears in storage as "packed" decimal. One byte contains two digits with the low-order byte containing the low-order digit followed by the sign of the item. For items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration interpreted as positive but which does not represent an overpunch.

BINARY ITEM: A binary item may be considered decimally as consisting of numeric characters 0 through 9 plus a sign. It occupies two bytes (a half-word), four bytes (a full-word), or eight bytes (two words), corresponding to specified decimal lengths of 1 to 4 digits, 5 to 9 digits, and 10 to 18 digits, respectively. The leftmost bit of the reserved area is the operational sign.

If the item is used as a resultant data name in an arithmetic statement, and no ON SIZE ERROR option is specified, the area may be set to a number greater than that specified in the PICTURE clause.

If the item is used as an operand, it is assumed that the area contains a number less than or equal to that specified in the PICTURE clause.

An Example of binary represented decimal:

If the number is +1234 and,

```
Picture and Usage Are:              Machine Representation Is:

PICTURE S9999 COMPUTATIONAL.        |0 000|0100|1101|0010|
                                     S          \_________/

                                                   BYTE
```

S=Sign a "1" in position S means number is negative.

a "0" in position S means number is positive.

PROCEDURE DIVISION

The Procedure Division of a source program specifies those procedures needed to solve a given problem. These steps (computations, logical decisions, input/output, etc.) are expressed in meaningful statements, similar to English, which employ the concept of verbs to denote actions, statements and sentences to describe procedures. The Procedure Division must begin in Area A with the header PROCEDURE DIVISION followed by a period.

The discussion that follows describes the units of expression that constitute the Procedure Division and the way in which they may be combined.

Its constituent parts, in order of hierarchy, are:

- Section
- Paragraph
- Sentence
- Expression
- Statement

SECTIONS

A section is composed of one or more successive paragraphs and must begin with a section-header beginning in Margin A. A section-header consists of a unique section-name conforming to the rules for procedure-name formation, followed by the word SECTION and a period. A section header must appear on a line by itself, except in the Declaratives portion of the Procedure Division, where it may only be followed immediately by a USE sentence or an INCLUDE statement. The INCLUDE statement is discussed in Section 7. A section-name need not immediately follow the words PROCEDURE DIVISION or END DECLARATIVES. A section ends at the next section-name or at the end of the Procedure Division, or, in the case of Declaratives, at the next section-name or at END DECLARATIVES.

PARAGRAPHS

A paragraph is a logical entity consisting of one or more sentences. Each paragraph must begin with a paragraph-name starting in Margin A.

A paragraph-name must not be duplicated within the same section. When used as operands in Procedure Division statements, non-unique paragraph-names may be uniquely qualified by writing IN or OF after the paragraph-name, followed by the name of the section in which the paragraph is contained. A paragraph ends at the next paragraph-name or section-name, or at the end of the Procedure Division. In the case of Declaratives, a paragraph ends at the next paragraph-name, section-name, or at END DECLARATIVES.

SENTENCES

A sentence is a single statement or a series of statements terminated by a period and followed by a space. A single comma or semicolon or the word THEN may be used as a separator between statements. A sentence must be contained within Margin B.

EXPRESSIONS

An expression may be defined as a meaningful combination of names, literals, COBOL words, and/or operators which may be reduced to a single value. This definition will become clear after the reader has studied the two types of expressions employed in COBOL, the "arithmetic" expression and the "conditional" expression.

STATEMENTS

A statement consists of a COBOL verb or the word IF or ON, followed by any appropriate operands (data-names, file-names, or literals) and other COBOL words that are necessary for the completion of the statement. The three types of statements are: compiler-directing, imperative, and conditional.

Types of Statements

COMPILER-DIRECTING STATEMENT: A compiler-directing statement directs the compiler to take certain actions at compilation time. A compiler-statement contains one of the compiler-directing verbs and its operands. Compiler-directing statements (except for NOTE, COPY, and INCLUDE) must appear as separate single sentences.

IMPERATIVE STATEMENT: An imperative statement specifies an unconditional action to be taken by the object program. An imperative statement consists of a COBOL verb and its operands, excluding the Compiler-Directing verbs and the conditional statements. An imperative statement may also consist of a series of imperative statements.

CONDITIONAL STATEMENT: A conditional statement is a statement containing a condition that is tested to determine which of alternate paths of program flow to take.

The following rules apply to the arithmetic statements:

1. All data-names used in arithmetic statements must represent elementary numeric data items that are defined in the Data Division of the program, except when they are the operands of the GIVING option.
   Operands of the GIVING option can be either elementary numeric or report.
2. The maximum size of any data-name or literal is 18 decimal digits.
3. Intermediate result fields generated for the evaluation of fixed-point arithmetic expressions assure the accuracy of the result field, except where high order truncation is necessary.
4. Decimal point alignment is supplied automatically throughout computations.

The ROUNDED and SIZE ERROR options apply to all the arithmetic statements. The GIVING option applies to all arithmetic statements but COMPUTE.

GIVING Option: If the GIVING option is written, the value of the data-name that follows the word GIVING will be made equal to the calculated result of the arithmetic operation. The data-name that follows GIVING is not used in the computation and may contain editing symbols.

If the GIVING option is not written, the operand following the words TO, FROM, BY, and INTO in the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements, respectively, must be a data-name. This data-name is used in the computation and is made equal to the result.

ROUNDED Option: If, after decimal-point alignment, the number of places in the calculated result are greater than the number of places associated with the data-name whose value is to be set equal to the calculated result, truncation occurs unless the ROUNDED option has been specified.

When the ROUNDED option is specified, the least significant digit of the resultant data-name has its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

Rounding of a computed negative result is performed by rounding the absolute value of the computed result and then making the final result negative (unless the final result is zero).

SIZE ERROR Option: Whenever the number of integral places in the calculated result exceeds the number of integral places specified for the resultant data-name, a size error condition arises.

If the SIZE ERROR option has been specified and a size error condition arises, the value of the resultant data-name is not altered and the series of imperative statements specified for the condition is executed.

If the SIZE ERROR option has not been specified and a size error condition arises, no assumption should be made about the final result; but the program flow is not interrupted.

It should be noted that the SIZE ERROR option applies only to final calculated results. When a size error occurs in the handling of intermediate results, no assumption should be made about the final result.

An arithmetic statement, if written with a SIZE ERROR option, is not an imperative statement. Rather, it is a conditional statement and is prohibited in contexts where only imperative statements are allowed.

Table

Permissible Comparisons

First Operand \ Second Operand	GR	AL	AN	ANE	NE	FC* NNL	ZR NL	ED	BI	ID	EF	IF	SR	SN	IN	IDI
Group (GR)	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN		
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Alphanumeric Edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Numeric Edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Figurative Constant (FC)* & Nonumeric Literal (NNL)	NN	NN	NN	NN	NN			NN			NN		NN	NN		
Fig. Constant ZERO (ZR) & Numeric Literal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU	NU	NN	NU	IO[1]	
External Decimal (ED)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU	IO[1]	
Binary (BI)	NN						NU	NU	NU	NU	NU	NU		NU	IO[1]	
Internal Decimal (ID)	NN						NU	NU	NU	NU	NU	NU		NU	IO[1]	
External Floating Point (EF)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU		
Internal Floating Point (IF)	NN						NU	NU	NU	NU	NU	NU		NU		
Sterling Report (SR)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Sterling Nonreport (SN)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU		
Index Name (IN)							IO[1]	IO[1]	IO[1]	IO[1]					IO	IV
Index Data Item (IDI)															IV	IV

*FC includes all Figurative Constants except ZERO

[1] Valid only if the numeric item is an integer

Notes:

NN	comparison as described for nonnumeric operands
NU	comparison as described for numeric operands
IO	comparison as described for two index-names
IV	comparison as described for index data items

APPENDIX D

ENTRIES FOR VARIOUS MANUFACTURERS

As was noted in Chapter 1, one of the goals of ANSI COBOL was to make the COBOL programming language machine-independent, that is, make it so that a COBOL program could be compiled and executed on any piece of hardware from any manufacturer without change.

The only exception to this goal was supposed to have been in the ENVIRONMENT DIVISION, where specifications were to be contained for each machine on which the programs were to be compiled and executed. It is useful, therefore, to be aware of some of the entries which may be made in the Environment Division for various manufacturers and Operating Systems which are available. It should be noted, however, that these differences, unfortunately, are not the only ones where there are variances from machine to machine. The programs and examples presented in the text use entries which should be machine independent within the Data Division and the Procedure Division. The reader should be cautioned, however, that some differences between the formats and statements illustrated in this text and those used with some compilers may differ. Thus, if some errors occur within a program, either diagnostic or execution errors, the programmer may find it prudent to check the manufacturer's manuals for particular uses of the statement causing the error.

The following is a compilation of some of the Environment Division entries which may be used for different machines. This list is not all inclusive and it may be well to augment the list with manufacturer's publications.

Basic Format

```
ENVIROMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. source-computer-entry
OBJECT-COMPUTER. object-computer-entry
[SPECIAL-NAMES. special-names-entry]
[INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT Clause
   ASSIGN Clause
   RESERVE Clause]
```

IBM System/360 and System/370

$\underline{\text{SOURCE COMPUTER}}$. computer-name.
$\underline{\text{OBJECT COMPUTER}}$. computer-name.

System/360		System/370 Model 165		System/370 Model 155	
Code Letter	Capacity	Code Letter	Capacity	Code Letter	Capacity
C	8K	I	512K	H	256K
D	16K	J	1024K	HG	348K
E	32K	JI	1536K	I	512K
F	65K	K	2048K	IH	768K
G	131K	KJ	3072K	J	1024K
H	262K			JI	1536K
I	524K			K	2048K
J	1048K				

$\underline{\text{SELECT}}$ file-name $\underline{\text{ASSIGN}}$ TO system-name

DOS

SYSnnn-class-device-organization [-name]

OS

class [-device]-organization-name

GE Series 6000/600

$$\underline{\text{SOURCE-COMPUTER}}.\quad \left\{\begin{array}{l}\underline{\text{6000}}\\ \underline{\text{GE-6000 LINE}}\\ \underline{\text{GE-625}}\\ \underline{\text{GE-635}}\end{array}\right\}.$$

$$\underline{\text{OBJECT COMPUTER}}.\quad \left\{\begin{array}{l}\underline{\text{6000}}\\ \underline{\text{GE-600 LINE}}\\ \underline{\text{GE-625}}\\ \underline{\text{GE-635}}\end{array}\right\}.$$

$$\underline{\text{ASSIGN}}\text{ TO file-code-1 }\left[\text{FOR }\left\{\begin{array}{l}\underline{\text{CARDS}}\\ \underline{\text{LISTING}}\end{array}\right\}\right]$$

Burroughs B2500 and B3500

$$\underline{\text{SOURCE COMPUTER}}.\quad \left\{\begin{array}{l}\underline{\text{B-2500}}\\ \underline{\text{B-3500}}\end{array}\right\}\ .$$

$$\underline{\text{OBJECT COMPUTER}}.\quad \left\{\begin{array}{l}\underline{\text{B-2500}}\\ \underline{\text{B-3500}}\end{array}\right\}\ .$$

ASSIGN TO hardware-name-1

Some of the allowable hardware-names are:

PRINTER
PUNCH
READER

TAPE (7 or 9 channel tape)
TAPE-7 (7 channel tape)
TAPE-9 (9 channel tape)

CDC 3170, 3300, and 3500

$$\underline{\text{SOURCE COMPUTER}}.\quad \left\{\begin{array}{l}\underline{\text{3170}}\\ \underline{\text{3300}}\\ \underline{\text{3500}}\end{array}\right\}\ .$$

$$\underline{\text{OBJECT COMPUTER}}.\quad \left\{\begin{array}{l}\underline{\text{3170}}\\ \underline{\text{3300}}\\ \underline{\text{3500}}\end{array}\right\}\ .$$

ASSIGN TO hardware-name-1 dis-1

Allowable hardware-names are:

READER
PUNCH
PRINTER
TAPE

DISK
SYSTEM-INPUT
SYSTEM-OUTPUT
SYSTEM-PUNCH

UNIVAC 1100 Series

SOURCE-COMPUTER. UNIVAC-1108.
OBJECT-COMPUTER. UNIVAC-1108.

ASSIGN TO implementer-name [system-file-name]

The implementer-name may be one of the following:

CARD-READER
CARD-PUNCH
PRINTER
FORM-PRINTER
UNISERVO
MASS-STORAGE

IBM 1130

SELECT file-name ASSIGN TO system-name

System Names for Card and Printer Devices on 1130

RD-1442	for a 1442 Model 6 or 7 used for reading only.
PU-1442	for a 1442 Model 6 or 7 used for punching only.
RP-1442	for a 1442 Model 6 or 7 used for reading and punching.
PO-1442	for a 1442 Model 5 (punching only).
RD-2501	for 2501 card reader.
PR-1132	for the 1132 printer when no carriage control is used.
PR-1132-C	for the 1132 printer when carriage control is used.
PR-1403	for the 1403 printer when no carriage control is used.
PR-1403-C	for the 1403 printer when carriage control is used.

JOB STREAMS

In Chapter 12, the Sort program is used to sort data into a prescribed sequence. On most computer systems, this requires entries in the job stream to define the files to be used with the Sort program. The job streams which could possibly be used with IBM Disk Operating System and IBM Operating System are illustrated below.

DOS

```
// JOB SORT
// OPTION LINK
// EXEC FCOBOL

   - SOURCE DECK -

/*
// EXEC LNKEDT
// ASSGN  SYS016,X‘132’
// ASSGN  SYS001,X‘132’
// DLBL SORTED,‘SORTED U-P CDS’,71/001,SD
// EXTENT  SYS016,111111,1,0,3200,20
// DLBL  SORTWK1,‘SORT WORK’,71/001,SD
// EXTENT SYS001,111111,1,0,3240,40
// EXEC

   - DATA -

/*
/&
```

OS

```
// SORIT  JOB  - - -
// STEP1  EXEC  COBULG
// COB.SYSIN  DD  *

   -  SOURCE DECK  -

// GO.OUTPUT1     DD SYSOUT=A
// GO.SORTWK01    DD UNIT=SYSDA,SPACE=(TRK,(200),,CONTIG)
// GO.SORTWK02    DD UNIT=SYSDA,SPACE=(TRK,(200),,CONTIG)
// GO.SORTWK03    DD UNIT=SYSDA,SPACE=(TRK,(200),,CONTIG)
// GO.SORTWK04    DD UNIT=SYSDA,SPACE=(TRK,(200),,CONTIG)
// GO.SORTOUT     DD UNIT=SYSDA,SPACE=(80,20),DISP=(,PASS), DSN = &SYSUT3
// GO.INPUT1      DD *

   -  TEST DATA  -

/*
//
```

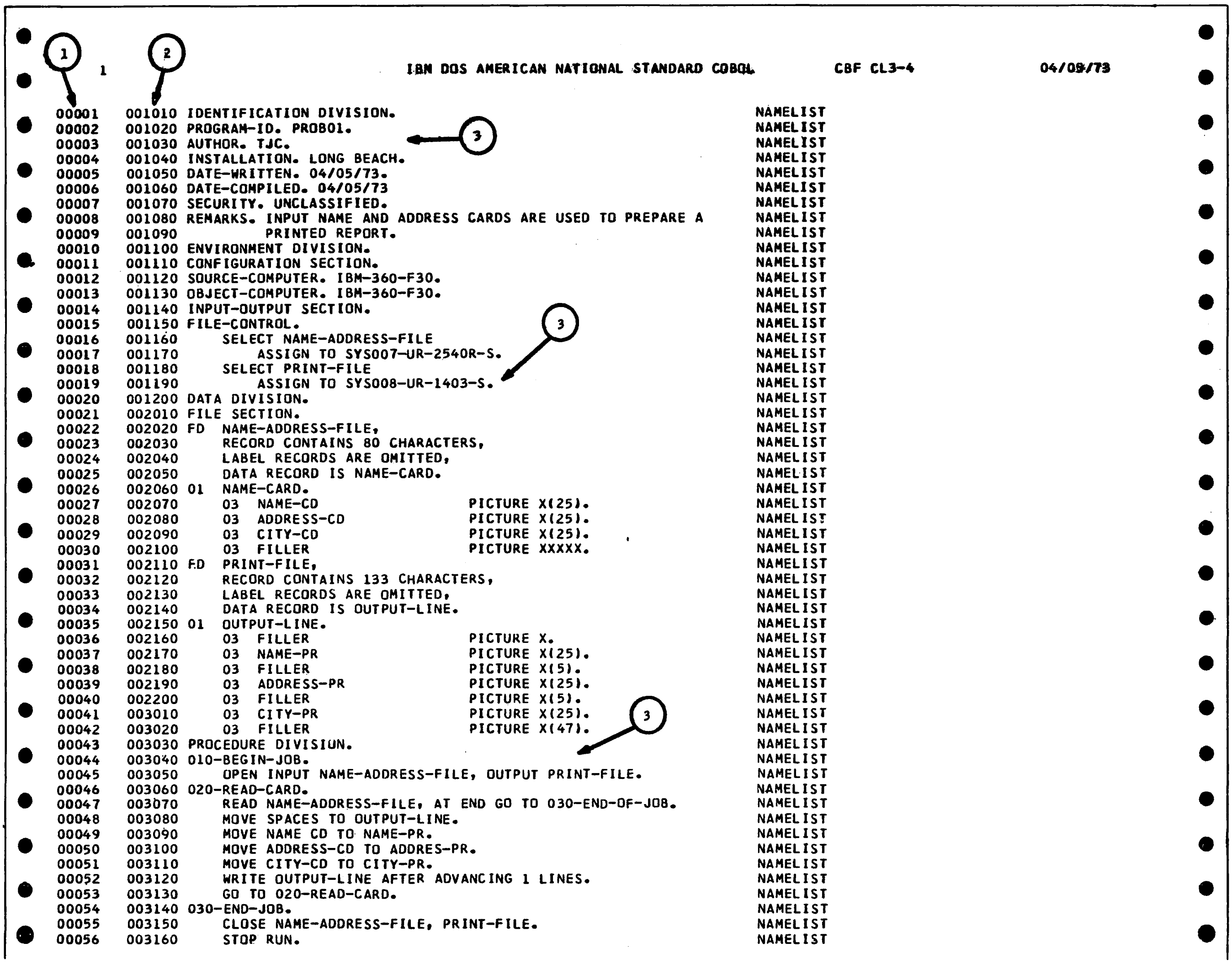
APPENDIX E

READING SOURCE LISTING AND DIAGNOSTICS

The compiler and related software of the computer system used for ANSI COBOL can produce output in the form of printed listings and/or punched card object modules. In addition, the printed listings may contain "diagnostics" and informative messages when errors are detected in the source program. An explanation of the contents of the listing is given below.

SOURCE LISTING

Figure E-1 is an example of a source module listing from the IBM DOS ANSI compiler.



```
     1                                  IBM DOS AMERICAN NATIONAL STANDARD COBOL         CBF CL3-4              04/09/73

00001   001010 IDENTIFICATION DIVISION.                                         NAMELIST
00002   001020 PROGRAM-ID. PROB01.                                              NAMELIST
00003   001030 AUTHOR. TJC.                                                     NAMELIST
00004   001040 INSTALLATION. LONG BEACH.                                        NAMELIST
00005   001050 DATE-WRITTEN. 04/05/73.                                          NAMELIST
00006   001060 DATE-COMPILED. 04/05/73                                          NAMELIST
00007   001070 SECURITY. UNCLASSIFIED.                                          NAMELIST
00008   001080 REMARKS. INPUT NAME AND ADDRESS CARDS ARE USED TO PREPARE A      NAMELIST
00009   001090          PRINTED REPORT.                                         NAMELIST
00010   001100 ENVIRONMENT DIVISION.                                            NAMELIST
00011   001110 CONFIGURATION SECTION.                                           NAMELIST
00012   001120 SOURCE-COMPUTER. IBM-360-F30.                                    NAMELIST
00013   001130 OBJECT-COMPUTER. IBM-360-F30.                                    NAMELIST
00014   001140 INPUT-OUTPUT SECTION.                                            NAMELIST
00015   001150 FILE-CONTROL.                                                    NAMELIST
00016   001160     SELECT NAME-ADDRESS-FILE                                     NAMELIST
00017   001170         ASSIGN TO SYS007-UR-2540R-S.                             NAMELIST
00018   001180     SELECT PRINT-FILE                                            NAMELIST
00019   001190         ASSIGN TO SYS008-UR-1403-S.                              NAMELIST
00020   001200 DATA DIVISION.                                                   NAMELIST
00021   002010 FILE SECTION.                                                    NAMELIST
00022   002020 FD  NAME-ADDRESS-FILE,                                           NAMELIST
00023   002030     RECORD CONTAINS 80 CHARACTERS,                               NAMELIST
00024   002040     LABEL RECORDS ARE OMITTED,                                   NAMELIST
00025   002050     DATA RECORD IS NAME-CARD.                                    NAMELIST
00026   002060 01  NAME-CARD.                                                   NAMELIST
00027   002070     03  NAME-CD                 PICTURE X(25).                   NAMELIST
00028   002080     03  ADDRESS-CD              PICTURE X(25).                   NAMELIST
00029   002090     03  CITY-CD                 PICTURE X(25).                   NAMELIST
00030   002100     03  FILLER                  PICTURE XXXXX.                   NAMELIST
00031   002110 FD  PRINT-FILE,                                                  NAMELIST
00032   002120     RECORD CONTAINS 133 CHARACTERS,                              NAMELIST
00033   002130     LABEL RECORDS ARE OMITTED,                                   NAMELIST
00034   002140     DATA RECORD IS OUTPUT-LINE.                                  NAMELIST
00035   002150 01  OUTPUT-LINE.                                                 NAMELIST
00036   002160     03  FILLER                  PICTURE X.                       NAMELIST
00037   002170     03  NAME-PR                 PICTURE X(25).                   NAMELIST
00038   002180     03  FILLER                  PICTURE X(5).                    NAMELIST
00039   002190     03  ADDRESS-PR              PICTURE X(25).                   NAMELIST
00040   002200     03  FILLER                  PICTURE X(5).                    NAMELIST
00041   003010     03  CITY-PR                 PICTURE X(25).                   NAMELIST
00042   003020     03  FILLER                  PICTURE X(47).                   NAMELIST
00043   003030 PROCEDURE DIVISIUN.                                              NAMELIST
00044   003040 010-BEGIN-JOB.                                                   NAMELIST
00045   003050     OPEN INPUT NAME-ADDRESS-FILE, OUTPUT PRINT-FILE.             NAMELIST
00046   003060 020-READ-CARD.                                                   NAMELIST
00047   003070     READ NAME-ADDRESS-FILE, AT END GO TO 030-END-OF-JOB.         NAMELIST
00048   003080     MOVE SPACES TO OUTPUT-LINE.                                  NAMELIST
00049   003090     MOVE NAME CD TO NAME-PR.                                     NAMELIST
00050   003100     MOVE ADDRESS-CD TO ADDRES-PR.                                NAMELIST
00051   003110     MOVE CITY-CD TO CITY-PR.                                     NAMELIST
00052   003120     WRITE OUTPUT-LINE AFTER ADVANCING 1 LINES.                   NAMELIST
00053   003130     GO TO 020-READ-CARD.                                         NAMELIST
00054   003140 030-END-JOB.                                                     NAMELIST
00055   003150     CLOSE NAME-ADDRESS-FILE, PRINT-FILE.                         NAMELIST
00056   003160     STOP RUN.                                                    NAMELIST
```

Figure E-1 Source Module Listing

The source listing consists of:

1. LINE NO – A compiler-generated line number is printed in the leftmost column.

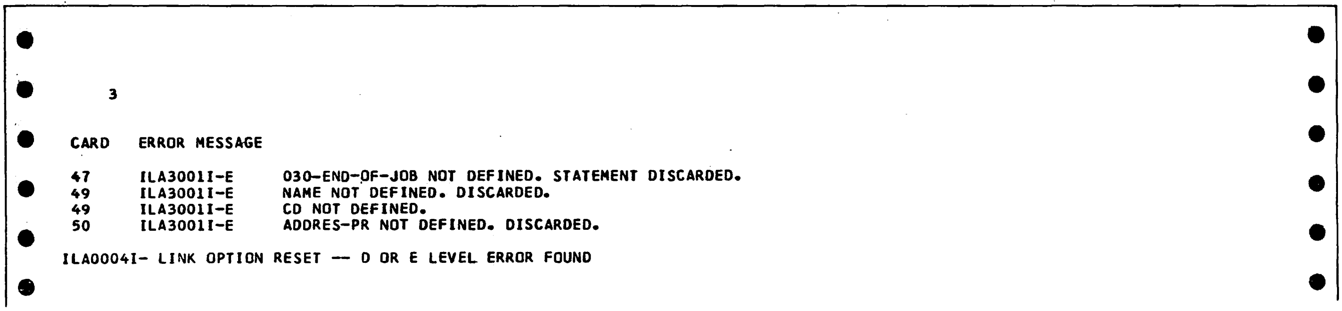
2. SEQUENCE NUMBER – The programmer provides the statement sequence numbers. They appear in the second column if punched on the COBOL Source Program.

3. SOURCE STATEMENT – All COBOL words and punctuation as keypunched in the Source Program are printed.

4. SEQUENCE ERROR (*) – If the sequence numbers are punched in the source deck, and one of the numbers is out of sequence, an asterisk will appear beside the source statement out of sequence. This is not illustrated in Figure E-1. Depending upon the compiler, a sequence number may or may not have significance.

DIAGNOSTIC MESSAGES

The segment of the listing in Figure E-2 is an example of a list of error messages. These diagnostic messages were generated by the DOS ANSI COBOL compiler for the program shown in Figure E-1. It should be noted that the format of the diagnostics and the diagnostic identification numbers will vary dependent upon the compiler and machine used.



```
       3


 CARD   ERROR MESSAGE

 47     ILA3001I-E     030-END-OF-JOB NOT DEFINED. STATEMENT DISCARDED.
 49     ILA3001I-E     NAME NOT DEFINED. DISCARDED.
 49     ILA3001I-E     CD NOT DEFINED.
 50     ILA3001I-E     ADDRES-PR NOT DEFINED. DISCARDED.

ILA0004I- LINK OPTION RESET -- D OR E LEVEL ERROR FOUND
```

Figure E-2 Diagnostics

1. CARD – This value contains the internal line number of the source statement where the error was detected.

2. ERROR MESSAGE – This line contains a message number and the severity of the error.

MESSAGE NUMBER	All message numbers and the associated messages are described in the appropriate programmer's guide.

Severity Code	Explanation
W = WARNING	This code indicates that an error was made in the source program. However, it is not serious enough to hinder the execution of the program.
C = CONDITIONAL	This indicates that an error was made but the compiler usually makes a corrective assumption. The statement containing the error is retained. Execution can be attempted for the debugging value.
E = ERROR	This indicates that a serious error was made. Usually the compiler makes no corrective assumption. The statement containing the error is dropped. Execution of the program should not be attempted.
D = DISASTER	This indicates that a serious error was made. Compilation is not completed and the results are unpredictable.

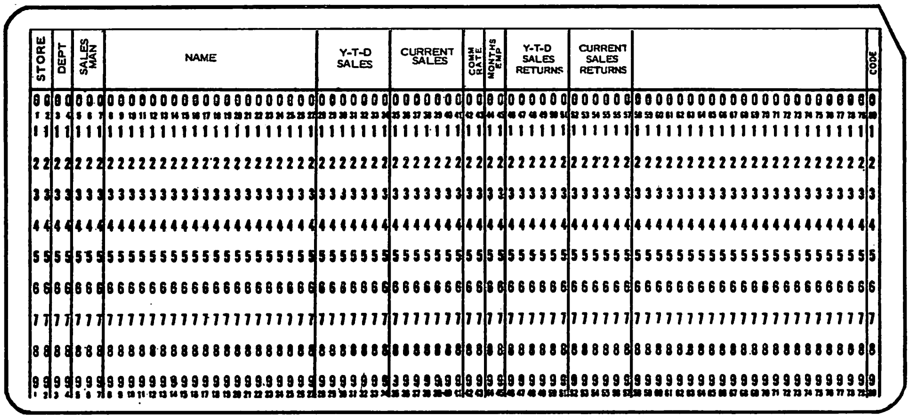
3. MESSAGE – Following the message number and the code is a sentence which indicates the nature of the error.

APPENDIX F

TEST DATA – PROGRAMMING ASSIGNMENTS

The following is an illustration of the complete card format for the input data cards used in the programming assignments presented throughout the text. Many programming assignments use this set of data cards.

CARD FORMAT - TEST DATA


STORE
DEPT
SALES MAN
NAME
Y-T-D SALES
CURRENT SALES
COMM RATE
MONTHS EMP
Y-T-D SALES RETURNS
CURRENT SALES RETURNS
CODE

Suggested test data is illustrated on the following page.

```
 STORE    DEPT    SLSMN                           YEAR TO DATE   CURRENT    COMM   MONTHS     YEAR TO DATE      CURRENT  CODE
 NUMBER   NUMBER  NUMBER        NAME                   SALES      SALES      RATE  EMPLOYED   SALES RETURNS  SALES RETURNS

   10      01      004    ACHER, WILLIAM C.          0190726    0057524     10      03         000850          000000     2
   10      04      027    ALHOUER, ELAINE E.         0112573    0022066     12      05         001880          000400     1
   10      03      030    ALLOREN, RUTH W.           2132020    0000000     15      10         114000          020933     2
   20      06      100    BATES, TONY F.             0819066    0207645     05      04         088340          005170     1
   20      08      102    BELLSLEY, ARTHUR A.        0883000    0099000     15      09         019033          002762     2
   10      09      105    BOYLE, RALPH P.            0878044    0143055     12      06         016070          007787     1
   10      02      111    CARTOLER, VIOLET B.        0280998    0075006     12      04         003020          001140     2
   20      05      122    CENNA, DICK L.             0355777    0044001     10      08         003270          000000     1
   10      04      171    COSTA, NAN S.              0580356    0056002     12      10         013020          000903     2
   20      06      179    DAMSON, ERIC C.            0352502    0180888     05      02         000223          000223     2
   10      03      181    DELBERT,EDWARD D.          1250659    0130554     15      09         037540          000000     2
   10      01      185    DONNEMAN, THOMAS M.        0650423    0090019     10      07         006700          001020     1
   20      05      207    EBERHARDT, RON G.          0564007    0094009     10      06         006300          000980     1
   10      07      214    EDMONSON, RICK T.          0079067    0033057     10      03         000290          000000     1
   10      09      215    EDSON, WILBUR S.           0607050    0082005     12      07         011885          001575     2
   20      08      282    ESTABAN, JUAN L.           1984055    0000000     15      10         040505          005050     2
   10      06      292    EVERLEY, DONNA M.          0332000    0177500     05      02         000322          000322     2
   20      01      300    FELDMAN, MIKE R.           0250000    0030000     10      09         000600          000000     2
   10      02      304    FROMM, STEVE V.            0650000    0120000     12      05         006090          001823     2
   10      05      308    GLEASON, JAMES E.          0145000    0039000     10      04         000925          000000     3
   10      07      310    GORMALLY, MARIE N.         0389022    0064055     10      06         007320          000866     1
   20      03      311    GROLER, GRACE B.           2300643    0205420     15      10         260480          048369     1
   20      09      315    HALE, ALAN A.              1274000    0168000     12      08         025954          003829     2
   20      04      317    HANBEE, ALETA O.           0385000    0039500     12      10         007530          001180     2
   20      03      318    HANEY, CAROL S.            0975000    0145000     15      05         012020          006019     3
   10      08      322    HARLETON, JEAN H.          0780899    0120089     15      06         010180          000000     2
   20      01      325    HATFIELD, MARK I.          0151122    0020539     10      07         001840          000220     3
   20      07      332    HELD, ANNA J.              0244000    0029500     10      09         005115          000926     3
   10      06      409    ICK, MICK W.               0410122    0195080     05      02         000590          000590     1
   10      03      487    KING, MILDRED J.           1850896    0180429     15      10         038260          005322     1
   10      08      505    LAMBERT, JERRY O.          0154001    0040745     15      04         002840          000000     5
   10      05      568    LYNNE, GERALD H.           0924487    0133311     10      07         016530          001428     1
   10      09      574    MELTZ, FRANK K.            0754366    0059030     12      10         015760          000000     1
   20      02      590    NEIL,CLARENCE N.           0750065    0095023     12      08         014000          002324     2
   10      06      607    ODELLE, NICHOLAS P.        0582507    0187507     05      03         006220          006220     3
   20      07      689    OWNEY, REED M.             0437788    0053066     10      09         008635          001132     1
   20      04      721    RASSMUSEN, JOHN J.         0810000    0100000     12      08         017680          002346     1
   10      01      730    REEDE, OWEN W.             0900555    0105144     10      08         010520          002310     1
   20      04      739    RIDEL, ROBERT R.           0557504    0082571     12      07         033303          007224     3
   20      09      740    RIDGEFIELD, SUZY S.        1804178    0190506     12      10         033024          002310     1
   20      02      801    SCHEIBER, HARRY T.         0095237    0032508     12      03         000520          000520     3
   20      07      802    SHEA, MICHAEL H.           0642033    0082009     10      08         012920          004680     1
   20      04      806    STUCKTON, NORMAN Q.        0672507    0072588     12      09         013380          013380     1
   10      09      820    TELLER, STEPHEN U.         1957044    0177009     12      10         051920          032222     3
   10      06      825    TILLMAN, DON M.            1230444    0199509     05      05         019230          004665     1
   20      03      834    TRAWLEY, HARRIS T.         0577326    0055000     15      10         011000          001329     2
   10      05      909    UDSON, DORIS M.            0154499    0030325     10      04         000780          000000     1
   10      08      921    ULL, GEORGE A.             1802000    0200000     15      09         034090          012080     3
   20      02      956    WANGLEY, THEO. A.          0120000    0015000     12      07         001720          000275     2
   10      01      960    WINGLAND, KEITH E.         0050000    0035000     10      02         000000          000000     1

TOTAL CARDS   50
```

INDEX

ACCEPT Statement, 13.1
ADD Statement, 3.4
  Giving Option, 3.5
  "TO" Option, 3.7
After Advancing Clause, 2.38
Alphabetic Moves, 2.35
ALTER Statement, 13.9
AND Operator, 6.6
ANSI COBOL, 1.8
Area A, 2.9
Area B, 2.9
Arithmetic Expression Evaluation, 5.4
Arithmetic Operators, 5.2
ASCENDING Clause, 12.11
ASCENDING KEY Clause, 12.16
Assembler Language, 1.4
ASSIGN TO Clause, 2.15
Asterisk Insertion, 3.13
AUTHOR Entry, 2.10

Basic Programming Concepts, 1.1
BLOCK CONTAINS Clause, 12.8
Blocked Records, 12.8

CALL Statement, 10.9
Carriage Control Tape, 8.2
Class Entry, 2.16
Class Test, 6.13
CLOSE Statement, 2.40
COBOL, 1.7
COBOL – Advantages, 1.12
COBOL – Disadvantages, 1.13
COBOL Format Notation, B.1
COBOL Reference Summary, C.1
COBOL Verbs, 2.30
Coding Form, 2.8
Comma Editing, 3.14
Comparing, 6.1
Comparison of Non-Numeric Items, 6.10
Comparison of Numeric Items, 6.9
Compiler, 1.10
Compound Conditions, 6.7
Computational-3 Format, 5.14
COMPUTE Statement, 5.1
Condition-Names, 7.1
CONFIGURATION SECTION Statement, 2.13
Credit Symbol, 3.17

DATA DIVISION, 2.7
DATA RECORD Clause, 2.20
Date Card, 8.12
DATE-COMPILED Entry, 2.10
DATE-WRITTEN Entry, 2.10
Debit Symbol, 3.17
DESCENDING Clause, 12.11
Device Entry, 2.16
DISPLAY Statement, 5.6
DIVIDE Statement, 4.7
Dollar Sign Editing, 3.14

Editing, 3.12
ENVIRONMENT DIVISION, 2.13
EXAMINE Statement, 7.12, 13.6
External Name, 2.15

FD Entry, 2.18
Figurative Constants, 2.33
File-Control Paragraph, 2.14
FILE SECTION, 2.17
Final Totals, 5.8
Flowchart Symbols, 2.6
Format Notation – Explanation, 2.11
Fortran, 1.4

GIVING Clause, 12.13
GIVING Option, 3.5
GO TO . . . DEPENDING Statement, 7.8
GO TO Statement, 2.39

Headings, 8.1

Identification Division, 2.9
INDEXED BY Clause, 12.16
IF Statement, 6.2
Input Editing, 7.9
INPUT-OUTPUT SECTION, 2.14
INPUT PROCEDURE Clause, 12.11
Insertion Characters, 3.14
INSTALLATION Entry, 2.10
Internal Data Representation, A.1

JUSTIFIED RIGHT Clause, 13.10

LABEL RECORDS Clause, 2.19
Level Indicators, 2.22

Level 88 Entries, 7.2
Line Counting, 8.8
Logical Operation, 6.5

Machine Language, 1.2
Major, Intermediate, Minor Totals, 10.1
Minor Total Logic, 9.1
Minus Sign - Editing, 3.15
MOVE CORRESPONDING Statement, 13.13
MOVE Statement, 2.33
MOVE Statement - Editing, 3.19
Multiple-Card Layout Form, 1.15
Multi-Level Tables, 11.7
Multiplication, 4.1, 4.3
MULTIPLY Statement, 4.3

Name Qualification, 13.11
NOT Operator, 6.5
NOTE Statement, 10.11
Numeric Data, 3.3
Numeric Literals, 3.6

OBJECT-COMPUTER Statement, 2.13
Object Program, 1.10
OCCURS Clause, 11.4
OPEN Statement, 2.31
OR Operator, 6.6
Organization Entry, 2.16
OUTPUT PROCEDURE Clause, 12.14

Paragraph-Names, 2.29
PERFORM Statement, 8.9, 13.2
Period Editing, 3.14
PIC Clause, 2.24
PICTURE Clause, 2.24
PL/I, 1.6
Plus Sign - Editing, 3.15
Print Overflow, 8.1
Printer Spacing Chart, 1.15
Problem Analysis, 1.13
PROCEDURE DIVISION, 2.29
PROGRAM-ID-Entry, 2.10
Programmed Switch, 9.1
READ Statement, 2.32
READ Statement, 2.32
RECORD CONTAINS Clause, 2.18
Record Description, 2.21
REDEFINES Clause, 7.10
Relation Test, 6.4
RELEASE Verb, 12.13
REMARKS Statement, 2.10
RETURN Statement, 12.15
ROUNDED Clause, 4.5
Rounding, 4.1
RPG, 1.5

SD Entry, 12.9
SEARCH ALL Statement, 12.18
SEARCH Statement, 12.20
Sections, 12.12
SECURITY Entry, 2.10
SELECT Statement, 2.15
SET Statement, 12.17, 12.18
Sign ("S" Entry), 5.12
Sign Test, 6.12
Sign Error Option, 4.6
SORT Statement, 12.10
SOURCE-COMPUTER Statement, 2.13
Source Listing and Diagnostics, E.1
Source Program, 1.10
SPECIAL-NAMES Paragraph, 5.6, 8.6
STOP Statement, 2.41, 5.18
Sub-Programs, 10.8
Subscripts, 11.4
SUBTRACT Statement, 3.10
  From Option, 3.11
  Giving Option, 3.10
Symbolic Device, 2.15

Table Lookup, 11.1
Table Search, 11.16
Test Conditions, 6.4
Testing and Debugging, 1.17

USAGE Clause, 5.14
USING Clause, 12.14
Utility Entry, 12.6

VALUE Clause, 5.13

WHEN Clause, 12.19
WORKING-STORAGE SECTION, 5.10
WRITE . . . FROM Option, 5.17
WRITE Statement, 2.38

Zero Suppression, 3.12

ANSI COBOL SERIES

ANSI COBOL - INTRODUCTION TO COMPUTER PROGRAMMING
by Gary B. Shelly and Thomas J. Cashman

COBOL programming is taught by means of example programs beginning with basic input/output operations, then basic arithmetic calculations, comparing, control break logic, table lookup, the use of the sort and search verb and many other topics. Each concept is illustrated through the use of typical business programs using industry-accepted programming techniques. Two programming assignments are included with each chapter offering a complete teaching unit on COBOL programming.

ISBN 0-88236-103-1 380 pages

ANSI COBOL WORKBOOK TESTING AND DEBUGGING TECHNIQUES AND EXERCISES
by Gary B. Shelly and Thomas J. Cashman

The ANSI COBOL workbook is designed to supplement the widely used textbook "Introduction to Computer Programming ANSI COBOL". Exercises include the analysis of the Procedure Division Listing, Data Division Listing, and related core dumps from actual COBOL programs which contain common programming errors. Saves hours of computing time by requiring the student to analyze typical diagnostics occurring in COBOL programs before the errors occur.

ISBN 0-88236-104-X 258 pages

ADVANCED ANSI COBOL DISK/TAPE PROGRAMMING EFFICIENCIES
by Gary B. Shelly and Thomas J. Cashman

This advanced ANSI COBOL textbook is designed to be used as a text in the second semester, or quarter COBOL programming class. It is a sequel to the introductory ANSI COBOL textbook. This text has two major emphases: Familiarizing the student with the commonly found access methods for disk and tape programming, and also introducing the student to proper techniques in coding COBOL programs for efficient debugging and execution.

ISBN 0-88236-105-8 363 pages

DOS JOB CONTROL FOR COBOL PROGRAMMERS
by Gary B. Shelly and Thomas J. Cashman

This unique textbook includes job streams, examples, and detailed explanations of the following types of processing: execution of COBOL programs for link-and-go, cataloguing COBOL programs, job control for processing with magnetic tape, job control for magnetic disk and indexed sequential files, the DOS Library Directories, and service programs for the Core Image Library, the Relocatable Library, the Source Statement Library.

ISBN 0-88236-265-8 300 pages